ROBUST

Reliability of Basic and Ultra-reliable Software sysTems

An Integrated Software Reliability Tool

Yashwant Malaiya
Jason Denton
Naixin Li
{malaiya, denton}@cs.colostate.edu
http://www.cs.colostate.edu/ testing/robust

ROBUST is a tool for analyzing and predicting the growth of software reliability based on a number of software reliability growth models. It supports four traditional software reliability growth models, as well as Malaiya's Logarithmic Coverage Model. Capable of working with either time or failure interval data, ROBUST allows for a wide variety of enhancement techniques to be applied to the models it supports. ROBUST is capable of computing a number of metrics for its data sets, including the Laplace factor and the mean relative error. An early version of ROBUST is available for many Unix systems and for Microsoft Windows 95/NT. To obtain a copy for evaluation and testing, contact the authors at the address above.

ROBUST is the result of research conducted by Dr. Yashwant Malaiya, Dr. Naixin Li, and Jason Denton. It is supported by a BMDO funded project monitored by ONR.

Contents

Chapter 1

Getting Started with ROBUST

When you first run ROBUST two windows will appear, marked "Original Data" and "Model Data". Initially these two windows will be empty. To load a data file, select the menu option **File|Open**. This brings a up a standard system open dialog from which you can select the name of the file to be opened. By default ROBUST only displays files names ending in ".fds". Usually such data files contain failure time or interval data for use with traditional software reliability models. When a file is opened ROBUST places the data in the window marked "Original Data". ROBUST automatically fills in failure intensity for each data point, as well as either failure time or failure interval, depending on what field is missing from the data that was read in.

Once a data file has been loaded you can begin to create models based on it. When you select one of the models on the model menu a textual view of the generated model will appear in the window marked "Model Data". This view is similar to the one of the original data, it contains the information about the predicted failure count and failure intensity at each time point in the original data.

You can see a graph of your data at anytime by selecting the **Query|Graph** option from the menu. Doing so brings up two more windows. The first is a graph of time vs. failure intensity. The second is a graph of time vs. cumulative failures. These graphs will always show original data in green. The mode, if you have one selected, will be shown in blue. The parameters values used to generate the model, in any, will be shown at the bottom of the screen. Changes in enhancement techniques or the model applied will cause the graph to change to reflect the appropriate change.

Selecting **Query|Model Information** from the menu will open a window displaying information about the model. The dialog box displays the name of the model and the parameters used in fitting the model to the data. MRE, MPE, count bias, and intensity bias are all measures of the models accuracy. MRE stands for Mean Relative Error, and is a measure of long term predictive ability; MPE stands for Mean Predictive Error and is a measure of a model's short term predictive ability. In both cases lower scores are better.Count and intensity bias is the average bias of the model in predicting the failure count and intensity, respectively. The final entry, Laplace factor, is a measure of trend in the data. Values above 2 or below -2 a stable trend in the data.

ROBUST's predictive capabilities are accessible from the **Query** menu, by selecting one of the **Predict on...** entries. Selecting time, intensity, or failure count brings up a dialog box showing the appropriate piece of information for the last available data point. By entering a new value for this piece of information and clicking OK, you can see what values the current model predicts the

other two items will take one when the selected value reaches the specified value.

By using the **File|Revert** option on the menu you can clear any models you have generated, and revert back to the original data set; if you have employed any of the smoothing options explain in chapter 4 the results of this are also cleared. The **File|Reset** menu option will simply clear the current model and data set. Selecting the **File|Save** option will allow you to save the smoothed data set or model displayed in the current text window. When you are done using ROBUST select the **File|Exit** option from the menu to quit.

Chapter 2

Viewing and Working With Data

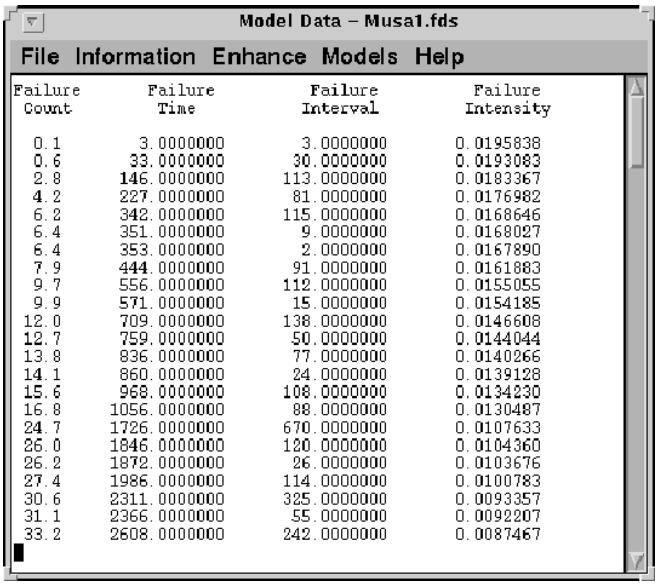
ROBUST deals with two primary forms of reliability data, and provides two primary means of viewing this data. ROBUST understands data about both failure times and intervals, and is also capable of producing models based upon coverage data. For both types of data robust provides two different ways of viewing both the original information and the models produced. The first method of display available with ROBUST is in text form, for easy inspection of individual data points. For getting an idea of model trends, ROBUST also supplies a graph of the data being modeled.

2.1 Text Views

By default, ROBUST displays information about data sets and models in a textual form in its two default windows, "Original Data" and "Model Data". The first window displays the original data collected by the user, the second contains the model selected by the user. The exact contents of these windows vary based on whether the data and models in use are based on failure times, or if they are based on coverage.

When ROBUST displays failure time data, either for a model or for an original data set, it displays four columns, giving four pieces of information for each data point. The first column is the failure count. The second column displays the time at which the failure occurred, measured from the start of testing. The third column is the failure interval, the elapsed time since the previous fault was found. For models these two models will match the information from the data set on which the model is based, with failure count and intensity adjusted to match model predictions. The final column is the failure intensity. For models, the failure time and interval columns will match those of the data set from which the model was created, with failure count and failure intensity adjusted to match model predictions at the given time step. A data set used by ROBUST contains either time or interval data, not both, ROBUST fills in the missing column when a new data set is loaded. Failure intensity for an original data set is computed as the increase in failure count divided by the increase in time between data points.

ROBUST displays coverage data and models is a somewhat different form. ROBUST accepts coverage data in the form of the number of faults encountered, and the percentage of an enumerable covered when that fault appears. An enumerable can be almost any measure part of a program, two common enumerables are program blocks and program branches. ROBUST allows for multiple enumerable coverage percentages to be supplied for each failure encountered. When displaying



Failure Count	Failure Time	Failure Interval	Failure Intensity
0.1	3.0000000	3.0000000	0.0195838
0.6	33.0000000	30.0000000	0.0193083
2.8	146.0000000	113.0000000	0.0183367
4.2	227.0000000	81.0000000	0.0176982
6.2	342.0000000	115.0000000	0.0168646
6.4	351.0000000	9.0000000	0.0168027
6.4	353.0000000	2.0000000	0.0167890
7.9	444.0000000	91.0000000	0.0161883
9.7	556.0000000	112.0000000	0.0155055
9.9	571.0000000	15.0000000	0.0154185
12.0	709.0000000	138.0000000	0.0146608
12.7	759.0000000	50.0000000	0.0144044
13.8	836.0000000	77.0000000	0.0140266
14.1	860.0000000	24.0000000	0.0139128
15.6	968.0000000	108.0000000	0.0134230
16.8	1056.0000000	88.0000000	0.0130487
24.7	1726.0000000	670.0000000	0.0107633
26.0	1846.0000000	120.0000000	0.0104360
26.2	1872.0000000	26.0000000	0.0103676
27.4	1986.0000000	114.0000000	0.0100783
30.6	2311.0000000	325.0000000	0.0093357
31.1	2366.0000000	55.0000000	0.0092207
33.2	2608.0000000	242.0000000	0.0087467

Figure 2.1: Text view of model data

coverage data ROBUST displays a series of columns; the first columns is the failure count and the remaining columns are the percentages of enumerables covered, in the order that they where specified in the coverage data file. Because models are based on only one enumerable, ROBUST displays only the failure count and the percentage of the enumerable in question when it displays models. Model displays always show one hundred data points, one for each one percent of the enumerable covered. The failure count for each point is computed based on Malaiya's logarithmic coverage model.

2.2 Graphs

One of ROBUST's most useful features is its ability to display data in a graphical format. The format of the information displayed is dependent upon what ROBUST is displaying in it's text window.

When working with traditional or static software reliability growth models two graphs are displayed. The first graph is a graph of failure intensity vs. time. The data upon ,which the model was based is shown in green, and the model itself is shown in blue. The second graph is a graph of cumulative failures vs. time. Again, original data is shown in green and the model in blue. Both graphs print the parameter values of the fitted model at the bottom of the display.

Adjusting the horizontal scroll bar to the right will cause the time scale on the bottom to increase by 10%, with the model extended to cover the new time period. Moving the scroll bar to the left will cause the time scale to decrease by 10%, with the view of the model contracted to show the failure data only up to the new time. Adjusting the vertical scroll bar will bring a horizontal line down across the graph. This line is an aid to the interpretation of the graph, the time entry at the bottom of the display shows the time where the horizontal line crosses the line of the model. The Intensity and Count entries at the bottom of the two displays show the failure intensity and

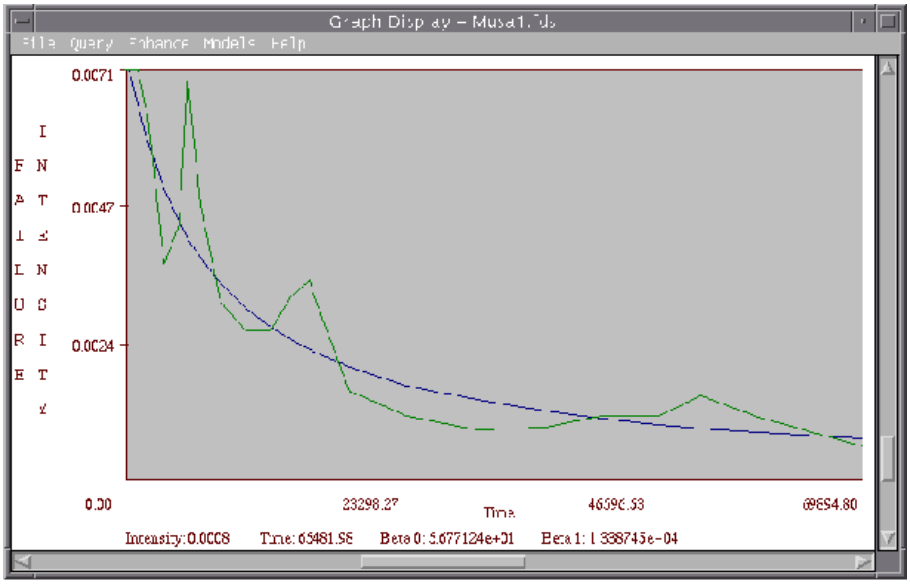


Figure 2.2: Graph view of model data

cumulative failures expected at the point where the model crosses the line.

When working with coverage data and Malaiya's Logarithmic Coverage Model, the previous two graphs are replaced with a single graph of enumerable coverage vs. faults found (or faults predicted). The horizontal scale (coverage) runs from 0 to 100% coverage, with the vertical scale running from 0 to the maximum number of faults found or predicted at appropriate. When no model has been generated, the graph will display plots of all enumerables present in the data set. When a model has been generated, the graph displays the data for the enumerable along side the model. Again, the model is in blue with original data in green.

Adjusting the horizontal scroll bar on a graph of a coverage model will cause a red cross to appear on the model line, and a line of text to appear at the bottom of the graph. As the scroll bar is adjusted this cross will follow the line of the model, and the text at the bottom will change to denote the percentage of the enumerable coverage marked by the cross, and the expected number of defects found at that level of coverage. This feature is only available when viewing a model, it does not work when inspecting graphical views of just coverage data.

2.3 Model Info

Selecting Model Info from the **Information** menu will bring up a dialog box containing basic information about the model you are currently working with. At the top of the model info dialog box is the type of model you are using. Underneath that are the parameters used to fit the model. For static failure models such as the Static Exponential model and the Static Logarithmic model, and for the coverage based models no additional relevant information is displayed.

For the traditional failure time models, four additional pieces of information are shown. The first is the Mean Relative Error (MRE) of the model. This is a measure of the models predictive accuracy. In theory, an MRE of 0% represents perfect predictive accuracy, with larger numbers

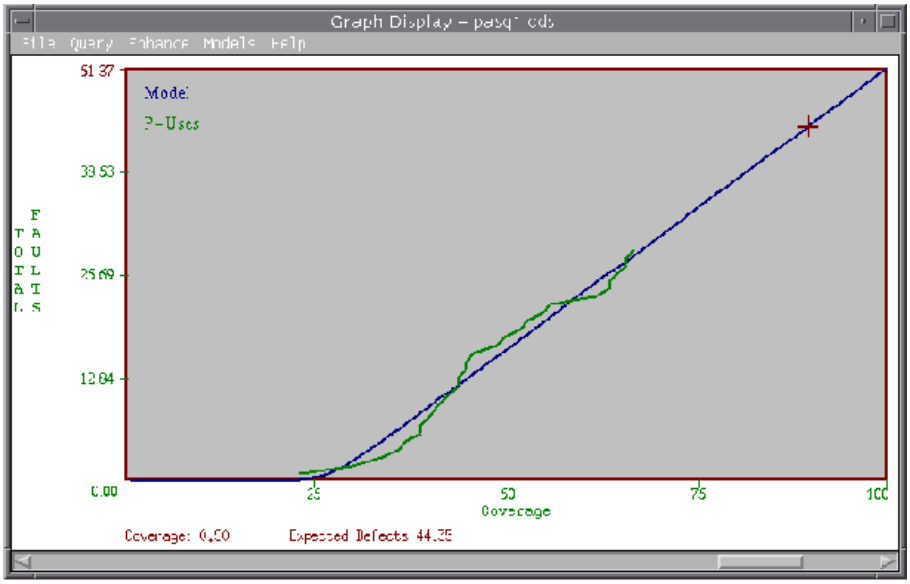


Figure 2.3: Coverage graph

representing less successful predictions. The second is a related term, the Mean Predictive accuracy. This is a measure of short term predictive accuracy, and again a score of zero indicates perfect accuracy. The entries marked count and intensity bias give the models average error in predicting the failure count and failure intensity, respectively. The final piece of information provided is the Laplace factor. This is a measure of trend present in the data. Positive values of the Laplace factor indicate decreasing reliability, while negative value of the Laplace factor indicate an increase in software reliability. Values above 1.96 or below -1.96 indicate that the data has developed a stable trend. The Laplace factor is always computed from the start of the data set.

2.4 Prediction

The bottom half of the Query menu is given over to three Predict on ... entries. Selecting one of these menu options will allow you to make predictions based on the current model. ROBUST allows the user to predict, and make predictions based on, failure intensity, failure count, and time.

Selecting the menu option corresponding to one of these three will display a dialog box. This dialog box shows what the latest prediction currently modeled is. For instance, if the model currently in use goes through to the 150th failure, the dialog box displays that information. The dialog box also displays a predict when prompt. This prompt varies with what is being predicting on. When the OK button is pushed another dialog box displays, showing you what the other two items are when the item you selected reaches the value entered by the user at the prompt.

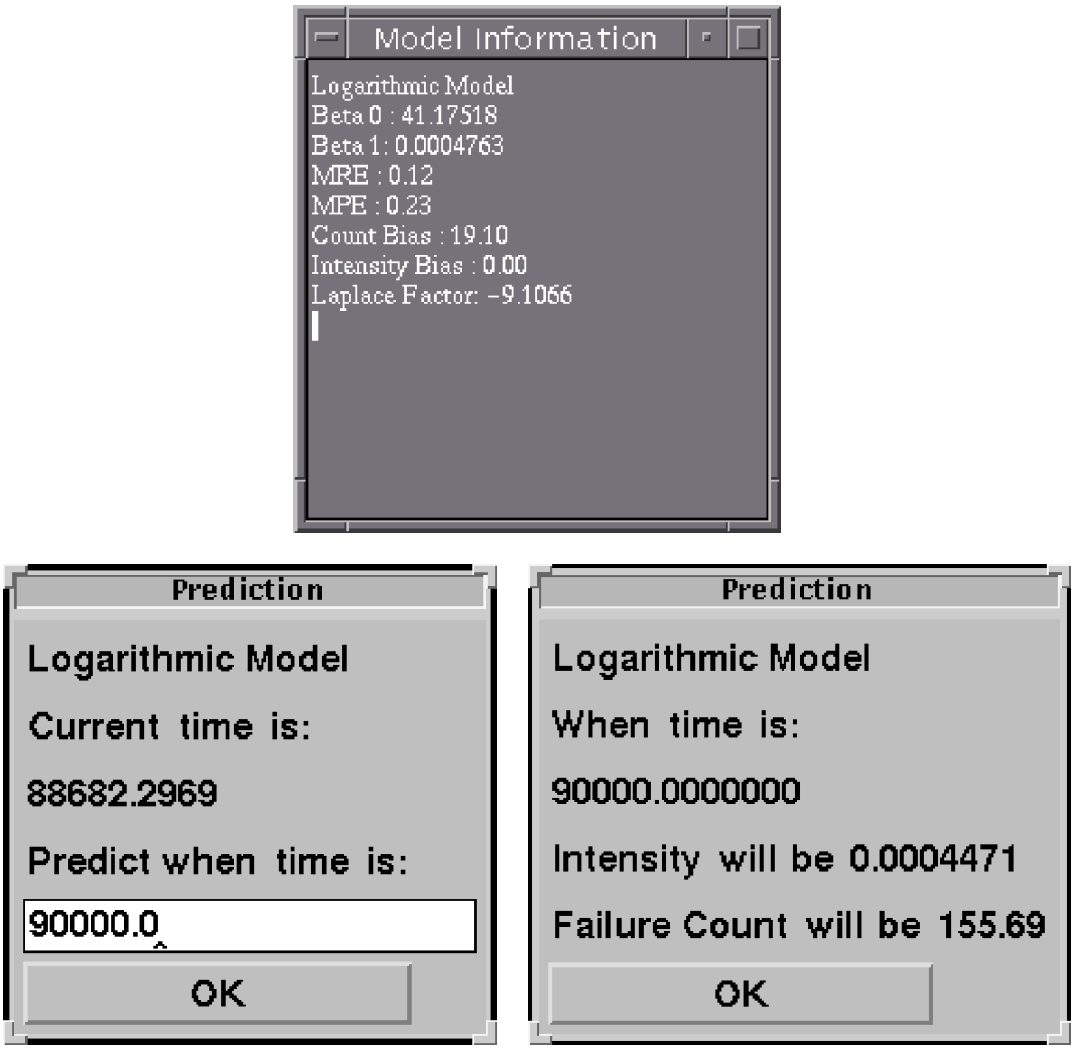


Figure 2.4: Prediction dialogs

2.5 Getting Help

ROBUST provides online help for many of the capabilities and ideas it implements. Selecting the menu item **Help|Help** will bring up a dialog box showing a list of possible help topics and two buttons. The first button, **Display Topic**, will cause a new window to open displaying the help text for the topic selected in the previous dialog box. The Help Topics dialog box will remain on the screen. To close it, select the second button, marked **Close**. Note that closing the help topics dialog box does not close the help window, and vice versa.

Chapter 3

Available Models

3.1 Traditional Models

ROBUST provides four traditional Software Reliability Growth Models. The table below shows what models are available, and what form they take.

Model Name	Mean Value Function	Failure Intensity
Exponential	$\beta_0[1-exp(-\beta_1 t)]$	$\beta_0\beta_1 e^{-\beta_1 t}$
Logarithmic	$\beta_0 ln(1+\beta_1 t)$	$\frac{\beta_0\beta_1}{1+\beta_1 t}$
Power	$\beta_0 t^{\beta_1}$	$\beta_0\beta_1 t^{\beta_1-1}$
Delayed S	$\beta_0[1-(1+\beta_1)e^{-\beta_1 t}]$	$\beta_0\beta_1^2 te^{-\beta_1 t}$

Selecting one of these models from the Models menu will generate the appropriate model using the original data provided, or the version of that data produced by one of the smoothing techniques if one has been applied.

3.2 Malaiya's Coverage Model

Dr. Li and Dr. Malaiya have proposed a software reliability model based on coverage an arbitrary enumerable, such as block coverage or branch coverage. This model is essentially a three parameter logarithmic model. It takes the form

$C_n^i = \alpha_0^i ln[1+\alpha_1^i(e^{\alpha_2^i C^i}-1)]$

where the α variables are the fitted parameters, and C^i is the percentage of the enumerable covered, and C_n^i is the expect number of faults.

Chapter 4

Enhancement Techniques

ROBUST allows for a variety of enhancement techniques to be applied to the various SRG models. It allows for two types of smoothing in addition to recalibration and stabilization. All enhancement techniques can be selected from the **Enhancement** menu option. Smoothing techniques are transformations applied to the original data set. Smoothing techniques can not be combined. Recalibration and Stabilization are techniques which are used after modeling, and are applied to the resulting model.

4.1 Smoothing

Part of the difficulty in obtaining an accurate fit for software reliability models comes from the fact that test data does not necessarily represent the true reliability of the software. This problem is particularly acute in the early stages of testing, when few data points are available and chance may place a number of data points far above or below the average, throwing off attempts to match a model to the data set. To compensate for this problem, several preprocessing techniques are supported by ROBUST. These techniques transform the data set, reducing the noise associated with the data set and allowing the model to be fitted more accurately.

As shown by Li and Malaiya the various smoothing techniques can result in dramatic gains for most data sets [2]. Although smoothing is not guaranteed to increase the accuracy of a given model for a given data set, they found that it rarely fails to do so.

When a smoothing techniques is applied, the data shown in the first text window is replaced with a smoothed version of the data. If a model is present it is re-applied to smoothed version of the data. From that point on all graphs and predictions will make use of the smoothed version of the data rather than the original. Note however that smoothing techniques are not cumulative, so selecting a second smoothing option from the enhancement menu will result in smoothed version of the original data set, not a smoothed version of the already smoothed data.

4.1.1 Fixed Grouping

One technique for smoothing which Naixin Li found to be effective was fixed sized grouping. Using this technique, one data point in the transformed data set represents a group of g data points in the original data set.

The fixed grouping algorithm selects the gth failure, the $2g$th failure, and so on. For each of the selected failures the algorithm sets the inter-failure interval to the median value for the group.

Although this technique has been found to be effective, it does suffer from the fact that it selects data points without regard to trends in the data. For this reason fixed grouping may lose some of the information in the data set. This danger grows with the size of group used. Small groups, however, provide less smoothing. Optimal group size may vary for different data sets, and there is no way to determine what optimal group size is for a given data set prior to completion of testing. Initial experiments by Naixin Li and Dr. Malaiya found good results where the data sets in question where reduced to fifty data points with grouping [2], later research suggested a group size of ten for most data sets. [5].

Note that this scheme differs somewhat from what is present in [2] and [5]. This is the result of continuing research and experimentation with filtering techniques.

4.1.2 Lump Grouping

Lump grouping corrects the fault with fixed grouping that the general trend of the data is not taken into account. Lump grouping operates on the assumption that when a failure occurs testing will be focused on correcting not only that fault but all similar faults as well. Thus failure intensity will then rise as these faults are detected and removed. Once these faults are removed, failure intensity will once again decrease until the next class of faults is uncovered.

Lump grouping makes use of this assumption by dropping all data points that are between two local minima. Formally, if $\lambda_i < \lambda_{i+1} < \lambda_{i+2} < ... < \lambda_j > \lambda_{j+1} > ... > \lambda_k$ then all points between $< u_i, t_i >$ and $< u_k, t_k >$ are dropped. This effectively ignores the temporary rise in failure intensity as a class of faults is corrected. This approach preserves data that can easily be lost by the fixed grouping method while still providing a degree of smoothing.

Lump grouping can be applied several times to a data set to increase the amount of smoothing. When Lump grouping is selected from the ROBUST menu a small dialog box appears which allows you to select the number of times you would like lump smoothing applied. Remember that robust uses the original data set for the input to all smoothing techniques. Should you decide you need to perform another iteration of lumping after having already smoothed the data, you should select a higher setting than what you used previously when asked how many times to perform lumping. Unfortunately lump grouping can quickly lose valuable data points, often rendering the data set to small to give accurate results, or even removing all but the first and last data points. Naixin Li and Dr. Malaiya suggest that lump grouping be applied to a data set twice provided that the resulting data set is large enough for analysis [5].

4.2 Recalibration

Most software reliability growth models exhibit some form of bias. To deal with this bias several techniques have been proposed. These techniques adjust the predictions made by a model based on the accuracy of previous predictions, and often result in an increase in the accuracy of the model.

ROBUST implements a simple form of recalibration found to be effective by Naixin Li. After a model is fitted to a data set, recalibration computes the average error that the model makes in

predicting failure intensity and failure count. These number are then added or subtracted from each data point as appropriate.

Recalibration is not full proof, and can reduce model accuracy in a few cases, usually when model accuracy was already very good. However, averaged over a large number of data sets, recalibration has been found to result in significant gains in model accuracy; making it a valuable technique for increasing the accuracy of a model.

When you select **Recalibration** from the **Enhancement** menu a small check will appear next to the menu option. From then on, whenever you fit a model to a data set the model will be recalibrated. You can turn recalibration off by selecting it on the menu a second time.

4.3 Stabilization

For some models it has been found that a combination of static and dynamic parameters can enhance predictive accuracy. Use of this technique requires that a clear interpretation of model parameters in terms of the development processes be available. For this reason stabilization is only applied to the exponential and logarithmic models. If stabilization is turned on then the parameters of these model are taken to be a combination of the parameters obtained through static estimation and dynamic fitting. In the case of the exponential model parameter β_0 is replaced with a static estimate when ever the dynamically fitted value exceeds one-half to two times the expected static value. In the case of the logarithmic model, stabilized parameter estimates are a weighted average between the static and dynamic estimates with the dynamic estimates getting twenty-five percent of the weight. This follows the recommendations in [9]

Chapter 5

Static Metrics

Often there exists a need to determine how much testing time is required before testing begins, or when testing is still in the early stages where reliable reliability data is not available. To assist with this problem ROBUST makes use of static metrics.

Static metrics are a collection of information about a program, the team developing that program, and the organization to which that team belongs. ROBUST posses the capability to generate generate reliability growth models based only on this information, and can also use this information to enhance predictions in the early phases of testing when reliability data is unreliable.

By selecting **Views|Static Metrics** from the menu you can bring up the static metrics dialog box. This dialog box allows you to adjust the parameters used in computing the static model. After you have adjusted the parameters pressing either the **Hide** or **Set** button will cause them to take effect. Pressing the **Hide** button will close the dialog box, the **Set** button will leave it on the screen.

The first five entries in the static metrics dialog box are Defect Density, Phase, Development Team, Maturity, and Structure. All of these entries are used to compute the number of initial defects, as explained in [6]. Phase refers to the phase of testing the program is currently in. The entry for development is the average skill level of the programmers working on the project. Maturity is the SEI maturity level of the developing organization. Structure is the percentage of the code that is done is assembly language, as opposed to some higher level language. Defect Density can be used as a calibrating factor, as explain in the section of the Static Models.

The remaining entries all relate to other aspects of the model. CPU rate is the speed of the processor used in testing, in MIPS. Src/Obj refers to the number of average number of object instructions generated for each line of source code. KLOC is the number of lines of source code, in thousands. The entry for time is used when static metrics is being used to predict reliability prior to testing. In this situation ROBUST generates 100 data points, spanning the length of time specified.

As explained later there are two forms for the logarithmic static model. One estimates the parameters used directly. When the direct estimation box is checked this is the method used. When direct estimation is not checked ROBUST estimates the parameters of the logarithmic model based on the parameters for the exponential model. The parameters Alpha defines the relationship between the two and is theorized to be in a relatively stable range for a given company or programming team. It is not used in direct estimation of the parameters, or when generating the parameters

of the exponential model.

5.1 The Static Models

Dr. Malaiya, Naixin Li, and Jason Denton have proposed that if the meanings of the parameters for the exponential and logarithmic models are known, then these parameters can be estimated from static information about the program, even before testing begins. In [6] they present a meaning for these parameters, and a method for estimating their value prior to the beginning of the test phase. ROBUST implements this scheme in the Static Exponential Model, and the Static Logarithmic Model.

These models take the same form as their parent models, the exponential and logarithmic model, respectively. Instead of fitting software reliability data, however, they estimate the parameters used based on data entered in the static metrics dialog box. If a data set is already loaded, ROBUST will generate a model which corresponds to the time and interval data for the original data set. If no data is available, ROBUST generates a model with 100 data points, evenly distributed over the period of time specified in the static metrics dialog box.

ROBUST has two methods for estimating the parameters of the logarithmic model. The first method, direct estimation, generates the model parameters directly from the available static data. The second method, indirect estimation, first estimates the parameters for the exponential model based on the static data, and then uses that estimate to estimate the parameters for the logarithmic model. This estimation is based on the parameter alpha, specified in the static metrics dialog box. It is theorized that this parameter is relatively stable for a given company or development team. It is recommended that you examine previous software projects in determining the value for this parameter in your organization, we suggest that it defaults to about 10.0. Which method ROBUST uses to estimate parameters is based on the Direct Estimation check box in the static metrics dialog box. When this box is checked, ROBUST makes use of the first method: direct estimation.

Chapter 6

Compiling ROBUST

Binaries for ROBUST are provided for a number of platforms. If your platform is not supported, it may still be possible for you to run ROBUST by down loading the source code and compiling it yourself.

ROBUST was written with the intent that it be portable to both Microsoft Windows and Unix environments. To enable this, ROBUST makes use of Bruce Wampler's V GUI toolkit. This toolkit it available from the V web site, at *http://www.objectcentral.com*. In order to successful compile ROBUST you must first download and compile the V libraries for the appropriate platform. The authors of ROBUST can not provide assistant with the task, but have found it to be a relatively easy task to get the V libraries running under a number of environments. If you need assistance in this, we recommend the V discussion mailing list. Information on this list can be found on the V homepage. ROBUST was developed with V version 1.15 and later, and will not work on with V versions 1.14 or earlier. Current builds user version 1.19.

6.1 For Unix

Building ROBUST for Unix requires making small changes to the makefile, dependent upon the platform and configuration for which ROBUST is being built. The most important change will be in selection of the appropriate platform. The beginning of makefile contains a number of selections for which ROBUST is known to work. Uncomment the appropriate selection. If your platform is not listed, you will have to configure the appropriate includes and X libraries inclusions yourself, most of the rest of the makefile will probably not be very helpful.

Next, select which release of the X11 system you want to use. It is recommended that you use release 6. The third available option controls how the program is compiled. Set CC to the compiler you want to use. This options defaults to g++, and it is recommend that you use this compiler. The FLAGS variable allows you to set additional flags of your own, but this should be unnecessary for most installations.

The next option, debug, should be set to either "yes" or "no". If set to yes, ROBUST will be built with debugging information included in the binaries. If set to no, ROBUST will be built using maximum optimization.

The final entry requiring user setting is the variable VBASE. This should point to the home directory of the V library. Note that this refers to the location of the directory named v, from which

the library was built, not the directory which actual contains the resulting archive. Once these all these entries are set appropriately, you should be able to build robust by simply running make.

6.2 For Microsoft Windows

Compiling ROBUST for Microsoft Windows should be a very straight forward process. Simply create new project in your favorite IDE, add all of the ROBUST *.cpp files to it, along with the V library, and hit compile. ROBUST should compile fine with either the Microsoft or Borland compiler, and most other compilers as well. Instructions for building the V library with various Windows compliers are available with the toolkit.

If using a gcc based compilier, setting the PLATFORM variable in the Makefile to windows should be enough to get ROBUST to build properly, see the Unix section for more details.

Appendix A

Common Questions and Problems

1. When running on a Unix system I get an error like: *ld.so.1: robust: fatal: libg++.so.2.7.2: can't open file: errno=2 Killed* **when I try to run the program**
This results because ROBUST is is written in C++. By default the GNU C++ complier makes used of dynamically linked libraries for certain C++ routines. Unfortunately, these are often not placed on the default search path. To fix this, adjust the environmental variable LD_LIBRARY_PATH to point to these libraries. Your system administrator should be able to help you.

2. ROBUST compiles properly, but crashes shortly after the program starts
This problem occurs because compliers for Intel based machines are capable of align data along byte, word, double word, or even quad word boundaries. Make sure that the V libraries and the ROBUST program where compiled using the same data alignment.

3. ROBUST crashes on startup
ROBUST expects the file robust.hlp to be in the same directory as it was started in. Make sure that this is the case. Usually this will mean running ROBUST from the directory where it was originally installed.

Appendix B

The FDS File Format

When a file is opened in ROBUST the open file dialog defaults to showing only those files which end in .fds, which stands for failure data set. This extension was chosen to help differentiate between files meant for ROBUST and those meant for other programs, ROBUST is itself not picky about the name of its input files.

ROBUST is picky about the contents of the file, however. Input files for ROBUST are plain ASCII text files containing a set of data points. Each data point consists of two real numbers. The first is the number of failures observed up to this data point. The second is either the time at which this failure occurred (time data) or the time since last failure (interval data). By default, ROBUST will assume that all data files contain time based data. This can be overridden by placing a data type specifier at the beginning of the input file.

A data type specifier is a string which appears as the first token of the input file. This string may be either “time” or “interval”, specifying that the following input of is of the respective type.

Example :

```
interval
1 3
2 30
3 113
4 81
5 115
```

Appendix C

The CDS file format

ROBUST supports a second file format for coverage data. By convention, coverage data is stored in files ending in ".cds", but robust is not concerned with the file name actually used. A coverage data file is a plain ASCII text file, with a header in ASCII format which specifies the contents and organization of the file.

A cds file starts with the key word "coverage". On the next line is the number of columns of data in the file. After this number is a series of keywords, specifying the word in which data in the file is entered. Legal keywords are: faults, blocks, branches, c_uses, and p_uses.

The remaining lines of the file are the actual data points, one data point to a line, with each data point consisting a series of numbers. Each number represents one entry, in the order given by the keywords in the header. For the entry corresponding to faults, this number should be an integer. For the remaining entries, ROBUST expects a series of percentages representing the percentage of each enumerable covered.

Example :

```
coverage
5 faults blocks branches c_uses p_uses
1 .34 .20 .26 .23
2 .42 .28 .34 .30
3 .48 .33 .40 .34
4 .54 .34 .44 .36
```

Appendix D

Command Line Interface

ROBUST supports a command line interface to facilitate research. The format for this command line interface is:
robust <filename> <option>
where <filename> is the name of the FDS or CDS file to be loaded when robust starts to execute, and option is one of the following:

log	Generate the logarithmic model
exp	Generate the exponential model
pow	Generate the power model
block	Generate the coverage model, based on block coverage
branch	Generate the coverage model, based on branch coverage
puse	Generate the coverage model, based on p-use coverage
cuse	Generate the coverage model, based on c-use coverage

The first three options work only for FDS files, the last four work only for CDS files. The appropriate coverage data must be available for the coverage model.

Bibliography

[1] J. D. Musa, A. Iannino, K. Okumoto, *Software Reliability - Measurement, Prediction, Applications*, McGraw-Hill, 1987.

A portion of the research that went into ROBUST is based on early work of Musa, and this book provides an invaluable reference and starting point into the field of software reliability.

[2] N. Li and Y. K. Malaiya "Enhancing Accuracy of Software Reliability Prediction" 4th international Symposium on Software Reliability, pp. 71-79, Nov. 1993

This paper details a number of methods that can be used to enhance software reliability models, and shows that the enhancement techniques used can be more important than the actual models. It is the original source for the smoothing techniques employed by ROBUST.

[3] Y. K. Malaiya, N. Li, J. Bieman, R. Karich and B. Skibble, "The Relationship Between Test Coverage and Reliability" 5th International Symposium on Software Reliability Engineering, Nov. 1994.

This is the first paper to present the coverage based software reliability model used by ROBUST. It provides background on the models bias is and development.

[4] N. Li and Y.K. Malaiya "ROBUST: A Next Generation Software Reliability Engineering Tool" Proc. IEEE Int. Symp. on Software Reliability Engineering, pp. 375-380, Oct. 1995

This is the first publication on ROBUST. It details a prototype of the system, which bears little relation to the existing system. It also describes a defect density model and a method of using statically estimated parameters to enhance predictive accuracy in the early phases of testing. Both of these features have been superceeded by recent research. The currently used defect density model can be found in [], the stabilization method described was since been shown to be of little value.

[5] N. Li, "Measurement and Enhancement of Software Reliability Through Testing," Ph.D. dissertation, Colorado State University, 1997.

The first prototype for ROBUST was implemented by Li to demonstrate work done towards his doctorate. His dissertation provides a compact and convenient source for the many of techniques which ROBUST employs, including coverage based reliability modeling, computation of the MRE. He also discusses fixed and lump grouping, and recalibration.

[6] Y. K. Malaiya and J. Denton "What do Software Reliability Parameters Mean?" IEEE Int. Symp. on Software Reliability Engineering, Nov 1997

This paper presents the defect density model used by ROBUST, and explains who it can be used to estimate the parameters of the logarithmic and exponential models.

[7] Y. K. Malaiya and J. Denton "Estimating Defect Density Using Test Coverage" Tech. Report, Colorado State University, 98-108.

This paper discusses the use of Malaiya's logarithmic coverage model in estimating the initial number of defects present in a program.

[8] Y. K. Malaiya, A. von Mayrhauser, and P.K. Sirimani "An Examination of Fault Exposure Ratio" IEEE Trans. Software Engineering, Nov 1993, pp 1087-1094.

This paper discusses the nature of the fault exposure ratio, and provides the foundation for some of the later work.

[9] J. A. Denton "Accurate Software Reliability Estimation", Masters Thesis. Colorado State University 1999.

This thesis presents the original work on stabilization, discusses the techniques that ROBUST implements to estimate parameters, and gives a good overview of the exponential and logarithmic models.