

THESIS
ACCURATE SOFTWARE RELIABILITY ESTIMATION

Submitted by
Jason Allen Denton
Computer Science

In partial fulfillment of the requirements
for the Degree of Master of Science
Colorado State University
Fort Collins, Colorado
Fall 1999

ABSTRACT OF THESIS

ACCURATE SOFTWARE RELIABILITY ESTIMATION

A large number of software reliability growth models are now available. It is widely known that none of these models performs well in all situations, and that choosing the appropriate model *a priori* is difficult. For this reason recent work has focused on how these models can be made more accurate, rather than trying to find a model which works in all cases. This includes various efforts at data filtering and recalibration, and an examination of the physical interpretation of model parameters. Here we examine the impact of the parameter estimation technique on model accuracy, and show that the maximum likelihood method provides for estimates which are more reliable than the least squares method. We present an interpretation of the parameters for the popular logarithmic model, and show that it may be possible to use this interpretation to overcome some of the difficulties found in working with early failure test data. We present a new software reliability model, based on the objective measure of program coverage, and show how it can be used to predict the number of defects in a program. We discuss the meaning of the parameters of this model, and suggest what needs to be done in order to gain a greater understanding of it. Finally, we present a tool we have developed which supports and integrates many of the techniques and methods presented here, making them easily accessible to practitioners.

Jason Allen Denton
Computer Science
Colorado State University
Fort Collins, Colorado 80523
Fall 1999

ACKNOWLEDGMENTS

I would like to thank Yashwant Malaiya for supporting my graduate schooling and research, and my friends and family for supporting me in my pursuit of higher education.

This research was supported in part by a BMDO funded AASERT project monitored by ONR.

TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	Introduction	1
1.2	Models to be Examined	2
1.2.1	The Exponential Model	2
1.2.2	The Logarithmic Model	2
1.3	Model Assessment	3
1.3.1	Short Term Predictive Capability: SRE	3
1.3.2	Long Term Predictive Capability: MRE	3
1.4	Data Sets Studied	4
1.5	Work on Enhancement of Software Reliability Growth Models	5
1.5.1	Lyu and Nikora’s Combination Models	5
1.5.2	Brocklehurst et al’s Recalibration	6
1.5.3	Schneidewind’s Optimal Data Selection	7
1.5.4	Li’s Work	7
1.5.4.1	Grouping	7
1.5.4.2	Recalibration	7
1.5.4.3	Stabilization	8
1.6	Conclusion	8
2	ESTIMATION OF PARAMETERS	9
2.1	Introduction	9
2.2	Least Squares Fitting on Failure Intensity	9
2.2.1	The Exponential Model	10
2.2.2	The Logarithmic Model	10
2.3	Maximum Likelihood Estimation	10
2.3.1	The Golden Section Routine	11
2.3.2	The Exponential Model	11
2.3.3	The Logarithmic Model	11
2.4	Comparison of Methods	12
2.5	Stability of Estimates	15
2.6	Conclusion	19
3	INTERPRETATION OF MODEL PARAMETERS	20
3.1	Introduction	20
3.2	Derivation of the Exponential Model	21
3.3	Implications of the Logarithmic model	22

3.3.1	Interpretation of the Logarithmic Model Parameters	23
3.4	Estimation of Logarithmic Model Parameters	24
3.4.1	Estimation through β_0^E and β_1^E	24
3.4.2	Direct Estimation of β_0^L and β_1^L	26
3.5	Concluding Remarks	26
4	STABILIZATION	27
4.1	Introduction	27
4.2	Estimation of Static Parameters	27
4.3	Stabilization Techniques	28
4.4	Experiments	30
4.4.1	Results From Initial Experiments	31
4.4.2	Analysis of Initial Experiments	38
4.4.3	Second Round of Experiments	38
4.5	Conclusion	43
5	A COVERAGE BASED MODEL	44
5.1	Introduction	44
5.2	A Coverage based approach	45
5.3	Applying the New Approach	46
5.4	Significance of Parameters	50
5.4.1	A Linear Approximation	51
5.5	Conclusions	52
6	ROBUST : AN INTEGRATED SOFTWARE RELIABILITY TOOL	53
6.1	Introduction	53
6.2	Capabilities	53
6.2.1	Traditional Software Reliability Models	53
6.2.2	Static Defect Density Model	54
6.2.3	Coverage Model Support	54
6.3	Development	56
6.3.1	A Java Version	57
6.4	Comparison to Other Tools	57
6.4.1	SMERFS	57
6.4.2	CASRE	57
6.5	The Future of ROBUST	58
7	CONCLUSION	59
	REFERENCES	61

LIST OF TABLES

1.1 Data Sets Used	5
2.1 Least Squares vs. Maximum Likelihood Estimation for the Exponential Model	13
2.2 LSQ vs. MLE for the Exponential Model, Smoothed Data	13
2.3 Least Squares vs. Maximum Likelihood Estimation for the Logarithmic Model	14
2.4 LSQ vs. MLE for the Logarithmic Model, Smoothed Data	14
2.5 Summary of Least Squares versus Maximum Likelihood	15
2.6 Standard Deviation of Parameter β_0	18
2.7 Standard Deviation of Parameter β_1	19
3.1 Comparison of model parameter interpretations	24
4.1 Static Parameters Used by Data Set, Known K	29
4.2 Results of Parameter Replacement Method for Exponential Model	32
4.3 Results of Parameter Replacement Method for Logarithmic Model, Direct Estimation	33
4.4 Results of Parameter Replacement Method for Logarithmic Model, Indirect Estimation	34
4.5 Weighted Average Method For The Exponential Model	35
4.6 Weighted Average Method For Logarithmic Model, Direct Estimation	36
4.7 Weighted Average Method For Logarithmic Model, Indirect Estimation	37
4.8 Static Parameters Used by Data Set, Estimated K	39
4.9 Results of Parameter Replacement with the Exponential Model, Estimated K	40
4.10 Weighted Average Method For The Exponential Model, Estimated K	41
4.11 Results of Parameter Replacement with the Logarithmic Model, Estimated K	41
4.12 Weighted Average Method For Logarithmic Model, Estimated K	42
4.13 Summary of Stabilization Results	42
5.1 Projected number of total defects with 100% coverage	49
5.2 Projected number of total defects with 100% coverage (Vouk's data sets)	50

LIST OF FIGURES

2.1	Parameter Stability for Exponential Model, Data Set T3	16
2.2	Parameter Stability for Logarithmic Model, Data Set T4	17
3.1	Variation of Fault Exposure Ratio with defect density	23
4.1	Linear and Non-Linear Stabilization Functions	30
4.2	Stabilization Step Functions	31
5.1	Estimated total defects using exponential model (Pasquini data)	47
5.2	Defects vs. Test Coverage Model	47
5.3	ROBUST: Block Coverage data (Pasquini et al.)	48
5.4	Defects vs. % Branch Coverage	48
5.5	Defects vs. % P-use Coverage	49
5.6	Defects vs. % P-use Coverage	50
6.1	Cumulative failure count vs. time for T1 data set	54
6.2	Failure intensity vs. time for T1 data set	55
6.3	ROBUST's Static Defect Density Model Dialog	55
6.4	A Graphical View of ROBUST's Coverage Model	56

Chapter 1

INTRODUCTION

1.1 Introduction

Increasingly software plays a critical part in not only scientific and business related enterprises, but in daily life where it runs devices such as cars, phones, and television sets. Although advances have been made towards the production of defect free software, any software required to operate reliably must still undergo extensive testing and debugging. This can be a costly and time consuming process, and managers require accurate information about how software reliability grows as a result of this process in order to effectively manage their budgets and projects.

The effects of this process, by which it is hoped software is made more reliable, can be modeled through the use of Software Reliability Growth Models, hereafter referred to as SRGMs. Ideally, these models provide a means of characterizing the development process and enable software reliability practitioners to make predictions about the expected future reliability of software under development. Such techniques allow managers to accurately allocate time, money, and human resources to a project, and assess when a piece of software has reached a point where it can be released with some level of confidence in its reliability. Unfortunately, these models are often inaccurate.

Numerous SRGMs have been proposed, and some appear to be better overall than others. Unfortunately, models that are good overall are not always the best choice for a particular data set, and it is not possible to know which model to use *a priori*. Even when an appropriate model is used, the predictions made by a model may still be less accurate than desired. For this reason, a great deal of research has gone into trying to make more effective use of existing models. Various methods have been proposed, such as adjusting for model bias (Brocklehurst, Chan, Littlewood, and Snell 1990; Li and Malaiya 1993; Li 1996), combining multiple models (Lyu and Nikora 1992b), and smoothing of initial data (Li and Malaiya 1993; Li 1996). Li and Malaiya found that the choice of improvement techniques often makes a bigger difference in model accuracy than the initial choice of model used (Li and Malaiya 1993).

This study will examine how existing software reliability growth models can be used more accurately. It will address holes in the existing literature, both in terms of model use and meaning, and suggest some techniques which might make software reliability models more accurate. In this chapter we will layout the basis for the work, models and data used, and techniques used to assess model effectiveness. We will also discuss prior work that has been done on the enhancement of these models. In chapter 2 we will examine the various techniques for estimating the parameters of SRGMs, and see how parameter estimation effects model accuracy. Chapter 3 presents work by Malaiya and Denton on a new interpretation of the parameters of the logarithmic model, and shows how this interpretation can be used to obtain estimates of the model parameters prior to the beginning of testing. Then, we will examine how these parameters can be combined with dynamic estimates from failure data to improve predictive accuracy in chapter 4. In chapter 5 we will present a new model based on program coverage, and then in chapter 6 we present the tool we have created to make our research available to practitioners. Finally, we will review our results and make recommendations for the accurate use of software reliability growth models in chapter 7.

1.2 Models to be Examined

Software reliability growth models are commonly designed to be used with data collected in terms of the testing time between failures, and it is on such models that this study will focus. These models are usually stated in terms of two equations; the mean value function and the failure intensity function. It is worth noting that the second function is the derivate of the first. The mean value and failure intensity functions are usually denoted as $\mu(t)$ and $\lambda(t)$, respectively. In this case t refers to the total time which has elapsed since the start of testing, as measured in seconds of CPU execution.

All of the models examined here have two parameters. Regardless of how these models were originally formulated, we will refer to the parameters of these models as β_0 and β_1 . When necessary, we will use a superscript to differentiate between parameters of different models. For example, β_0^E will refer to the β_0 parameter of the exponential model, and β_1^L will refer to the β_1 parameter of the logarithmic model. Standard practice is to determine the values of these parameters by fitting the model in question to the available data; we will examine the various means for doing so in chapter 2. Once the model has been fitted to the data, it can then be used to obtain estimates of current stability of the software and make predictions about the programs future reliability. The following sections describe the two models we will focus on and reference the papers in which these models were originally developed. Summaries of both these models are also available in (Farr 1996).

1.2.1 The Exponential Model

The most widely used software reliability growth model is the exponential model. This is a stochastic model based on a non-homogeneous Poisson process (Goel and Okumoto 1979). Originally proposed by Jelinski and Moranda in (Jelinski and Moranda 1971), many variations have since appeared. The original JM-exponential model made use of the elapsed wall clock time when a failure was encountered. A significant refinement was made by Musa, who restated the model in terms of CPU execution time allowing for more accurate predictions. Musa also described a method for moving between execution time and wall clock time, making it easier to make predictions in terms of real world calendars and deadlines (Musa 1975). Later, Goel and Okumoto worked to generalize the model, allowing the initial number of errors in a program to be random rather than fixed; and permitting errors to be independent (Goel and Okumoto 1979). Although superior to the earlier models, it has been shown that the exponential model is not generally the most accurate SRGM (Malaiya, Karunanithi, and Verma 1992). However, this model remains popular and widely used.

As previously stated, this study focuses on execution based models, measured in CPU seconds. For this case, the exponential model takes the form

$$\mu(t) = \beta_0[1 - e^{-\beta_1 t}] \quad (1.1)$$

$$\lambda(t) = \beta_0\beta_1 e^{-\beta_1 t} \quad (1.2)$$

where β_0 is taken to be the total number of defects present in the software, and β_1 is taken to be the per-fault hazard rate of the program. A detailed discussion of the model parameters is presented later, in chapter 3.

1.2.2 The Logarithmic Model

The logarithmic model was originally proposed by Musa and Okumoto in (Musa and Okumoto 1984). Like the exponential model, it models the failure process as a non-homogeneous Poisson process. The most significant difference between this model and the exponential is that the logarithmic model assumes that failure intensity will decrease exponentially with the expected number of failures experienced, while the exponential model assumes an equal reduction in failure intensity with each fault uncovered and corrected. In this sense it can be viewed as a continuous formulation of the geometric model (Musa and Okumoto 1984).

In (Malaiya, Karunanithi, and Verma 1992) it was shown that the logarithmic model was generally more accurate than many other SRGMs. It is relatively simple to use, although not as widely used as the exponential model. This may be due in part to the difficulty of obtaining a concrete interpretation of the model's parameters. We present such an interpretation in chapter 3.

The logarithmic model takes the form

$$\mu(t) = \beta_0 \ln(1 + \beta_1 t) \quad (1.3)$$

$$\lambda(t) = \frac{\beta_0 \beta_1}{1 + \beta_1 t} \quad (1.4)$$

1.3 Model Assessment

Our goal is to discover methods which will allow us to improve the predictive accuracy of software reliability growth models. To do this we need quantitative ways of measuring how accurate predictions made by these models are, for both the case where the original model is applied and the case where various enhancement techniques are used with a model. It is not enough to measure how well a model fits the observed data, we are not concerned with a model's ability to model past experience. Rather we are concerned with its ability to predict the future. In this study we make use of two measures of model accuracy, one to measure short term accuracy and another to measure the long term accuracy. Both of these accuracy measures are normalized with respect to the data sets they are measuring; we are examining the average error over all data points. This facilitates the comparison between raw data and data which has been smoothed. As discussed later we will be examining the effects of our proposals on both kinds of data and the smoothing process we use reduces the number of data points in a data set.

1.3.1 Short Term Predictive Capability: SRE

We will measure the short term predictive accuracy of software reliability growth models by observing how close they come to predicting the next observed failure. We will term our error measure Short term Relative Error, or SRE. To compute this we fit the model using only the first two data points, and then use the fitted parameters to predict how many errors will have occurred at the time when the third error was actually found (the third data point). We take the absolute value of the difference between these two values, and divide by the number of errors actually found at this time. We then repeat the process using the first three data points, then the first four, and so on until we have used the first $n - 1$ data points in the set to predict how many errors are expected at time t_n . We average the results over all predictions made.

Formally, we can state

$$SRE = \frac{1}{z} \sum_{i=2}^{n-1} \frac{|\mu_r(i+1) - \mu_p(i+1)|}{\mu_r(i+1)} \quad (1.5)$$

where $\mu_r(i+1)$ is the number of errors detected by time t_{i+1} and $\mu_p(i+1)$ is the number of errors the model predicts would be found by t_{i+1} when the first i data points are used fit the model.

The z term is used to obtain the average of all error predictions. Ideally, $z = n - 2$, indicating that we obtained a prediction error for all subsets of the data including the first two data points up to the next to last data point. Unfortunately, experience shows that in some cases it may not be possible to fit the parameters of a model to a data set for all data sets or subsets of data sets. This is particularly true when the data sets or sub-sets to be fitted are small. When this happens we can make no error measurement for the data set and model; z is the number of error predictions we were actually able to make for a data set and thus the number of terms in the summation.

1.3.2 Long Term Predictive Capability: MRE

Software reliability models are often employed to set schedules at the beginning of the test phase, making their long term predictive capabilities perhaps more important than their short term abilities. We will measure long term predicative accuracy in a manner similar to short term accuracy; using the Mean Relative Error or MRE. This error measure was first proposed by Li in (Li and Malaiya 1993).

The MRE of a model applied to a data set can be found by fitting the model parameters to the data using only the first two data points. Then the fitted model parameters are used to predict the number of errors that will be found by the end of testing. The absolute value of the difference between the predicted value and the actual number of faults found at the end of testing is then divided by the number of faults actually found to

find the error term. The process is repeated with the first three data points, the first four, and so on. All error terms are averaged over the number of error predictions made to find the final MRE.

Formally,

$$MRE = \frac{1}{z} \sum_{i=2}^n \frac{|N_0 - \mu_f(i)|}{N_0} \quad (1.6)$$

where N_0 is the total number of faults detected at the end of testing and $\mu_f(i)$ is the number of faults that the model predicts will have been found by the end of testing, based on fitting and adjusting the model based on the first i data points. Again, the z term is used to average over all predictions made when some subsets of the data will yield illegal parameter values; for the MRE $z = n - 1$ is the ideal case.

1.4 Data Sets Studied

All models and techniques examined here deal with data about the time at which failures occurred; or alternatively, data about the time between failure occurrences. These two forms can be considered equivalent. Although most software reliability growth models use data of this form, and such models have been in use for several decades, finding suitable data to verify models and improvement techniques is difficult. Early work generally focused on data based on calendar or wall clock time. Musa asserts that CPU execution time is a better measure than wall clock time, during which the actually time spent running a program can vary greatly based on CPU load, man hours, and other factors (Musa 1975). This study focuses on execution time, and so some early data is not applicable. To make matters worse, many companies are reluctant to release data about programs they have developed, so many new data sets rarely appear in the literature. As a result, new models are formulated and validated against data sets which can be published, or using data sets which have previously appeared elsewhere.

Despite these difficulties, data was available from two sources. The largest group of data comes from work done by Musa in the mid 1970's. These data sets, the T and SS series, are often used in the literature and represent some of the best available data. Unfortunately, the original source of these data sets, (Musa 1979), appears to no longer exist. Fortunately, the data sets themselves are available from the Department of Defense Data Analysis Center for Software, on their web site at <http://www.dacs.com/databases/sled/swrel.shtml>. Some auxiliary information about these data sets, such as the fault exposure ratio of the programs that generated this data, can be found in (Musa, Iannino, and Okumoto 1987). Care must be taken however, as there appears to be several naming schemes applied to this data, such that the same data set may be called by different names in different publications. Without the original report from which this data came it is difficult to determine why this is. As a result, this report will make use of the numbering provided by the DACS web site, assuming that the combination of software size and number of failures is enough to uniquely identify each data set, thus allowing data sets to be identified despite any inconsistencies in naming across publications. Data sets which are identified by only a number on the DACS web site, or by a number and single letter, will have a T pre-pended to them as this is the convention used by Musa in (Musa, Iannino, and Okumoto 1987). The SSx data sets will retain their names as specified on the web site.

Another source of data is Brocklehurst and Littlewood in (Brocklehurst and Littlewood 1996), from which comes the CSRx series of data. The first two data sets, CSR1 and CSR2, are mentioned in (Brocklehurst and Littlewood 1992) as being used to validate the work presented there. Unfortunately, Brocklehurst and Littlewood do not provide the details of these data sets in that paper. They expand on the work in (Brocklehurst and Littlewood 1992) and (Brocklehurst and Littlewood 1996), and the CD-ROM which accompanies this book contains the CSRx data sets. The CSR1 data set contains 397 user-perceived failures. These include usability problems and issues arising from poor documentation. The CSR2 data set contains a subset of the failures reported in CSR1, those which are known to be due to software faults. CSR3 purports to be another subset of CSR1, this time one containing only those failures related to Pascal Programming. Unfortunately, how this differs from CSR2 is not explained, but we will assume that all three provide reasonable data to work with. Table 1.1 lists the data sets to be used in this study, along with any auxiliary information about them which is available. This includes the number of defects in the data set, the number of machine level object instructions in the finished program, the type of application, and the fault exposure ratio K observed for the data in question.

Table 1.1: Data Sets Used

Data Set	Defects	Obj. Inst.(k)	Type	K($\times 10^{-7}$)
CSR1	397	Not Avail	Not Avail	Not Avail
CSR2	129	Not Avail	Not Avail	Not Avail
CSR3	104	Not Avail	Not Avail	Not Avail
T1	136	21.7	Real Time	1.87
T2	54	27.7	Real Time	2.15
T3	38	23.4	Real Time	4.11
T4	53	33.5	Real Time	10.6
T5	831	244.5	Real Time	4.2
T6	73	5.7	Commercial	3.97
T14C	36	Not Avail	Real Time	Not Avail
T17	38	61.9	Military	4.54
T40	101	180	Military	Not Avail
T27	41	126.1	Military	3.03
SS1A	112	Not Avail	OS	4.43
SS1B	375	Not Avail	OS	Not Avail
SS1C	277	Not Avail	OS	5.64
SS2	192	Not Avail	Time Sharing	Not Avail
SS3	278	Not Avail	Word Proc.	Not Avail
SS4	196	Not Avail	OS	1.41

1.5 Work on Enhancement of Software Reliability Growth Models

A large number of software reliability growth models now exist, but none has emerged as a clear choice for use in modeling the software development process. Although the two models presented here stand out as good overall choices, it has become apparent that all models are highly variable in terms of predictive ability, and there appears to be no method for choosing which model is best suited to a given project *a priori* (Abdel-Ghaly, Chan, and Littlewood 1986). For this reason substantial effort has been placed into developing techniques which improve the predictive accuracy of existing SRGMs. These techniques tend to take one of two approaches. The first approach recognizes that one of the main sources of inaccuracy in SRGMs is the instability of test data, particularly in the early phases of testing. These techniques focus on smoothing or filtering of the testing data to provide an improvement in accuracy. The second approach deals with the bias present in SRGMs, and makes attempts to compensate for this bias, either by recalibrating the predictions made by the model or by combining models with different biases together to form composite models.

It is unfortunate, but many researchers focused only on their own attempts at improving model accuracy, and made no attempt to evaluate how well their techniques worked in comparison to other techniques. Direct comparison is further complicated by the fact that each researcher used a different measure of model accuracy. Also, very few researchers examined how well their techniques might combine with others. This makes it hard to determine what are the best techniques. Throughout this study we will examine how our proposals interact with the data filtering proposed by Li, which he showed often improved model accuracy (Li and Malaiya 1996). We begin by reviewing the individual techniques.

1.5.1 Lyu and Nikora's Combination Models

One of the simplest methods for dealing with model bias is Lyu and Nikora's linear combination models (Lyu and Nikora 1992b). Lyu and Nikora recognized the fact that many models have an innate bias towards optimistic or pessimistic prediction. They suggest that this bias can be dealt with by combining models of opposite biases. Lyu and Nikora examined four methods of combining the logarithmic, exponential, and Littlewood-Verrall model. They compared their results against the three component models and three additional models and found that overall the various combination models they examined provided improved predictive accuracy. Although individual component models occasionally outperformed a combination model, other researchers have noted that there is no way of knowing *a priori* which models these may be (Abdel-Ghaly, Chan, and

Littlewood 1986).

Lyu and Nikora propose three different methods of combining component models to obtain a composite model. The first method proposed they term the Equally Weighted Linear Combination model, or ELC. Here, all three predictions are averaged to come up with the prediction made by the combined model. In the Unequally Weighted Linear Combination model, or ULC, the average is weighed in favor of the median estimate. The optimistic and pessimistic predictions each contribute one-sixth of their value to the final prediction, with the median prediction contributing four-sixths of its value. Note that the weights for the ULC are determined dynamically at each time step based on the values of all predictions, rather than set at the beginning of the modeling process based on what a model's bias is believed to be *a priori*. The third linear combination model is the Median-Oriented Linear Combination Model, or MLC. Here, the median prediction at each time step is taken as the prediction of the combination model, with the optimistic and pessimistic predictions dropped. Again, the determination of what is the optimistic and pessimistic prediction is made after each error is observed.

The final combination model examined by Lyu and Nikora is the Dynamically Weighted Linear Combination model, or DLC. Overall, they found that this was the best of the combination models. Unfortunately, Lyu and Nikora state that this model uses changes in the prequential likelihood function to assign model weights, but do not elaborate on how this is done. They do state that the basic DLC model examines the change in prequential likelihood over only one time step. Later in (Lyu and Nikora 1992b) they examine two variations on DLC where changes in the prequential likelihood over a larger time step are examined. With DLC/F they examine changes in the prequential likelihood over a fixed window, with the next window starting at the end of the previous one and model weights recomputed only at the end of the window. The DLC/S method uses a sliding window, with weights recomputed after every data point. Both of these extended versions of DLC compute the weights of their component models by dividing the change in the prequential likelihood function of each component by the change in the prequential likelihood function that changed the most. These weights are then normalized to sum to one. We assume that the basic DLC model uses a similar procedure to set its weight in the absence of a more detailed explanation.

Lyu and Nikora observe that their overall technique need not be limited to the three models they experimented with, but is easily extended to instead use those models that a particular users feels may be more appropriate. Alternatively, a combination model need not be limited to only three component models. It is important, however, to chose models with opposite biases.

1.5.2 Brocklehurst et al's Recalibration

In (Brocklehurst, Chan, Littlewood, and Snell 1990) Brocklehurst, Chan, Littlewood and Snell propose an alternative method for dealing with model bias. Their method, which they call recalibration, is based on the estimated distribution of the inter-failure times. This technique assumes that the error in predictions is essentially stationary, at least over the short term. They do suggest however, that this assumption need not always be meat for recalibration to yield an improvement in the predictive quality of the model.

They observe that the true distribution of inter-failure times, F_i , is related to the observed inter-failure times by some unknown function which they term G . They suggest that G may be estimated through the use of a *u-plot*, which is sample cumulative distribution function of the sequence of u_i values, where each u_i is the probability that the prediction of the next time to failure is smaller than the eventually observed value. By joining these u_i values on the plot, either with simple line segments between each value or a three-knot spline, they create a continuous estimate for G . Then, the recalibrated prediction of the failure interval distribution, \hat{F}_i , is $\hat{F}_i = G_i[F_i(t)]$.

Brocklehurst et al present data showing that this technique is applicable to any SRGM, and can greatly improve model performance even when the model itself performs poorly. This recalibration method has the advantage that it allows the prequential likelihood ratio to be used to decided whether or not recalibration is helping to make more accurate predictions. On the downside, this method of recalibration addresses only the distribution of failure intervals, and it is hard to see how this relates back to individual predictions, which is what interests managers. They also observe that their technique is designed for dealing only with the next prediction, and more work is required to verify or adapt it to making longer term predictions.

1.5.3 Schneidewind's Optimal Data Selection

Schneidewind addresses the problem of inaccurate reliability models by examining the data on which they are based (Schneidewind 1993). He suggests that the most recent failures provide better data about program reliability than do previous data points. Schneidewind examines the case where only data from some time s to the time testing ends is used to fit the model and make predictions. He finds that this can lead to substantial improvements in predictive capability.

In (Schneidewind 1993) Schneidewind focuses on the choice of s , examining several criterion for the selection of s . Generally, these are all numeric techniques, involving the maximization or minimization of various functions. All criterion showed an improvement in reliability prediction for some data sets, but none were found to be consistently superior. Schneidewind suggests that s be selected so the mean squared error between predictions and the actual number of cumulative failures on the interval $s \leq i \leq t$ is minimized. Schneidewind suggests this method based on the fact that it gives good results and is relatively easy to compute compared to some of the other techniques for finding s which he examined.

The shortcoming of Schneidewind's works is that it is developed only with relation to his own software reliability growth model. This model uses data in terms of the number of failures in a constant length interval, which is different from the other models examined in this study. While Schneidewind's ideas and work might be adaptable to models such as those studied here, doing so would be a major work by itself, and so Schneidewind's work is not considered in later sections. However, the idea of using only a portion of the original failure data is important, and represents an important step forward in improving the accuracy of software reliability growth models.

1.5.4 Li's Work

Li and Malaiya examined both methods of improving software reliability models; data filtering and adjusting for bias (Li and Malaiya 1993; Li 1996). Perhaps more importantly, they looked at how techniques to handle both problems could be combined. They found that although such techniques do not always offer an improvement in reliability, they did make a significant difference in many cases. Choice of enhancement techniques was often more important to software reliability growth modeling than initial model choice (Li 1996).

1.5.4.1 Grouping

Li and Malaiya addressed the issue of noise in software reliability data by applying several different smoothing techniques. The first technique, fixed grouping, reduces the original data points from $\langle \mu_1, t_1 \rangle, \langle \mu_2, t_2 \rangle, \dots$ to $\langle \mu_1, t_1 \rangle, \langle \mu_{g+1}, t_{g+1} \rangle, \langle \mu_{2g+1}, t_{2g+1} \rangle, \dots$, based on a user selected group size g . Smaller groups filter out less noise, but larger groups run the risk of losing too much information. They ran experiments to determine the optimal value of g and found that setting g so that the grouped data set contained about 10 data points was optimal for the logarithmic, exponential, and power models (Li 1996).

Fixed grouping suffers from its failure to take the characteristics of a particular data set into account. Lump grouping addresses this by pruning all data points in a set which are not local minima with respect to failure intensity (the inverse of the failure interval). The rationale behind this approach is that testing is rarely random, and that once one fault is found testers will focus on similar faults thereby increasing the failure intensity until such a time that all faults of that class are removed and the failure intensity drops back to lower levels than what was observed prior to the first fault being found. Lump smoothing can and often does result in a smoothed data set which itself still contains some local minima, and Li and Malaiya found the two iterations of the lump smoothing processes was often optimal. Overall this technique was found to be superior to fixed grouping. In general they also found that grouping made little difference for models which already did a good job of describing the data, but had a more dramatic effect on models which did not (Li and Malaiya 1995).

1.5.4.2 Recalibration

Li and Malaiya also address the problem of model bias with a technique they call recalibration (Li and Malaiya 1993; Li 1996). Like Brocklehurst's recalibration based on the u-plot, Li and Malaiya's technique examines past performance of a model on a data set, and attempts to correct future predictions based on the observed previous errors. Unlike Brocklehurst's method, they directly adjust the predictions made.

The results they present for recalibration are for the simple tactic of computing the average bias in all previous predictions, and subtracting that bias from the next prediction. In practice, this means fitting the model using only the first two data points, making a prediction, and then calculating the bias between the prediction and the actual observed values. This process repeats with the first three data points, then the first four, and so on, until all data points have been used. It should be noted that recalibration varies, dependent on whether short term predictions about the next failure are required, or whether prediction about the eventual long term reliability of a program is being performed. In both cases, biases should be computed based on the appropriate predictions, either short term or long term.

Li and Malaiya examine recalibration in the case of long term prediction, and find that overall it aids in the accurate use of software reliability models. They also find that it combines well with their data grouping techniques, reducing the effects of over grouping and allowing for a further increase in predictive accuracy; although by itself it is less effective than grouping or smoothing alone. They suggest that an optimal approach may be to use grouping or smoothing combined with recalibration. Although this may not be optimal for any given data set, it results in an improvement of predictive accuracy for most data sets.

1.5.4.3 Stabilization

In (Li and Malaiya 1995) Li and Malaiya suggested a third technique for creating more accurate software reliability models. They observe that initial failure data is often unstable and contains too few data points for application of grouping techniques. This can lead to inaccurate estimation of model parameters, and then to inaccurate predictions. They suggest that if concrete interpretations of model parameters are known, then model parameters can be estimated using information about the software development process rather than based on early and unreliable failure data. Although such static parameter estimates can not be expected to be as good as those produced from failure data late in the development cycle, they may be better than early estimates made from smaller sets of failure data. Li and Malaiya suggest that combining dynamically estimated parameter values with static estimates can produce yet more accurate parameter values, and thus more accurate predictions, early in the development phase. They term this technique stabilization.

Li and Malaiya put forward a model for estimating the number of defects present in a program, and present methods of using this estimate to make static estimates of the parameters for the exponential and logarithmic models in (Li and Malaiya 1995). They suggest combining dynamic and static parameter estimates using a simple weighted average. Unfortunately, they do not make any recommendations about how to set the weights, and provide no experimental evidence to back the suggestion up. This study develops the idea of stabilization further, examining how weights might be set and what other alternatives might produce good results.

1.6 Conclusion

Researchers have come to the conclusion that no software reliability model is accurate for all programs and data sets. Worse, it appears that an appropriate model can not be chosen *a priori* (Abdel-Ghaly, Chan, and Littlewood 1986). To remedy this, work has been done on improving the accuracy of existing models, and it has been found that the appropriate choice of improvement techniques is often more important than initial model selection.

Our goal is to develop a set of recommendations about the use of software reliability models. How the parameters should be estimated, how the data should be filtered and smoothed, and what other techniques should be employed to ensure that traditional software reliability growth models give the best results possible. In the process of doing this we will examine exactly what the parameters of these models mean. Finally, we will present a new type of software reliability growth, one which addresses some of the shortcomings of traditional models.

Chapter 2

ESTIMATION OF PARAMETERS

2.1 Introduction

One of the primary difficulties in using software reliability growth models is in the estimation of their parameters. For some models, it is possible to transform the models and data so that linear least squares estimates can be fitted based on the failure intensity function. This method has the benefit that it is conceptually simple and easy to implement. Computationally it requires very little processing, allowing models to be fitted quickly. Informal discussions with practitioners suggest that this method of parameter estimation is often used for these reasons.

An alternative is the maximum likelihood method which attempts to determine those parameter values that are most likely to have produced a given data set. This method generally requires that some equation or set of equations be minimized. Because these equations are non-linear, it is necessary to make use of computationally expensive fitting routines. The accuracy of these routines is subject to the initial values given for the parameters which the user is trying to fit, and a poor initial choice of the unknown parameters can lead to parameter estimates which are inaccurate. Unfortunately, although the literature discusses how the maximum likelihood equations are derived at great length, recommendations on how to use them to obtain accurate parameter estimates are conspicuously missing. This applies not only to which algorithms should be used to find the parameter values, but also to what parameter values should be used to start the search.

In this chapter we will lay out the details of fitting our subject reliability models using both the least squares fitting technique (LSQ) and the maximum likelihood equations (MLE). We will then compare the predictive accuracy of the models using these two methods, in order to make a recommendation about which technique should be used. We also consider the case of raw data versus data which has been subjected to two rounds of lump smoothing, as suggested by Li (Li and Malaiya 1995). This allows us to ensure that we are making our comparison using the best known techniques for obtaining accurate software reliability growth models.

Follenweider, Karcich, and Knafl have addressed the issue of parameter estimation in the context of composite models (Follenweider, Karcich, and Knafl 1993). They examined the maximum likelihood method and several different least squares approaches and enhancements using several of measures of predictive accuracy. Their results are inconclusive, and the authors observe that the relative worth of the different procedures often depends on the performance criteria applied. Unfortunately Follenweider, Karcich, and Knafl do not examine the effects of parameter estimation on individual models, or for a large number of data sets. This study will address these issues, examining the effects of parameter estimation on individual models over a number of data sets. We will pay particular attention to how the methods in question alter both short term and long term accuracy, and make recommendations about how to obtain the best predictions for both the total number of faults to be found and how many faults are likely to be found in the short term.

2.2 Least Squares Fitting on Failure Intensity

For the two models examined here, it is possible to transform the failure intensity function into a linear form. Doing so also involves manipulation of the input data. Our procedure then is to transform the data, use a least

squares fitting routine to obtain the slope and intercept for the line which best fits this data, and then solve for the model parameters.

The procedure for the best fit line to a set of data can be found in many math books. In general, the slope of the best fit line can be found to be

$$\frac{\sum_{i=1}^n (x_i - \bar{x})y_i}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

and the intercept can be found to be

$$\bar{y} - \frac{\sum_{i=1}^n (x_i - \bar{x})y_i}{\sum_{i=1}^n (x_i - \bar{x})^2} \bar{x}$$

where x_i refers to the i^{th} x value, which in our case will always be the time to the x^{th} failure. The failure intensity at the i^{th} data point is y_i , with \bar{x} and \bar{y} referring to the average time and average failure intensity, respectively.

2.2.1 The Exponential Model

Equation 1.2 gives the expected failure intensity for the exponential model at time t . We can linearize this equation, and fit the actually failure intensity data to obtain estimates of the model parameters. Stated in a linear form, equation 1.2 becomes

$$\ln(\lambda) = \ln(\beta_0^E \beta_1^E) - \beta_1^E t$$

To obtain parameter estimates we take the natural log of the failure intensity observed at each time t and fit a line to this data. We then obtain parameter estimates by solving for the parameters in terms of the slope and intercept. In this case, the slope is $-\beta_1^E$, and the intercept is equal to $\ln(\beta_0^E \beta_1^E)$.

2.2.2 The Logarithmic Model

The failure intensity function for the logarithm function, equation 1.4, can be restated as

$$\frac{1}{\lambda} = \frac{1 + \beta_1^L t}{\beta_0^L \beta_1^L} = \frac{1}{\beta_0^L \beta_1^L} + \frac{1}{\beta_0^L} t$$

This is clearly a linear form, with an intercept of $\frac{1}{\beta_0^L \beta_1^L}$ and a slope of $\frac{1}{\beta_0^L}$. Since this linear form gives the reciprocal of the failure intensity, we take the reciprocal of the original intensity and fit the model to the resulting data. We then take the resulting slope and intercept and solve for the parameters of the original data set.

2.3 Maximum Likelihood Estimation

In much of the literature the preferred method of obtaining parameter estimates is to use the maximum likelihood equations. Likelihood equations are derived from the model equations and the assumptions which underlie these equations. The parameters are then taken to be those values which maximize these likelihood functions. These values are found by taking the partial derivate of the likelihood function with respect to the model parameters, the maximum likelihood equations, and setting them to zero. Iterative routines are then used to solve these equations. Unfortunately, the SRGM literature is sadly lacking in advice on which iterative routines to use, and with what starting values. This is unfortunate because the accuracy of parameter estimates and thus the accuracy of the models themselves greatly depends on the ability of the iterative search methods used to overcome local minima and find good values for the parameters.

2.3.1 The Golden Section Routine

In our work we make use of a golden section search routine in one dimension. As explained below, this is possible because when maximum likelihood parameter estimates are employed β_0 is always a function of β_1 . The golden section routine minimizes an error function by bracketing the region where the minima is supposed to be, and dividing this space with a third point. The routine then iteratively probes this space and adjusts the three initial points to cover a smaller region. Eventually the three points come close to convergence and the routine terminates, outputting an estimate of the parameter to be found and the final value of the error function. This algorithm is fully described in Numerical Recipes in C (Press, Teukolsky, Vetterling, and Flannery 1988), and we use the `golden` function presented in that work to do the actual computation. We also make use of the `mnrbrak` routine to find the initial bracketing of the error function.

Our approach is to use the Golden Section routine to search the parameter space for the value of β_1 which causes the maximum likelihood equations to be as close as possible to zero. In all cases we chose our initial best guess bracket to be 4×10^{-4} to 5×10^{-4} . We have found that these values seem to provide results which are reasonably accurate.

2.3.2 The Exponential Model

In (Farr 1996) the likelihood equation for the parameters of the exponential model is given as

$$L(\beta_0^E, \beta_1^L | n) = (\beta_0^E)^n (\beta_1^E)^n \left[\prod_{i=1}^n e^{-\beta_1^E t_i} \right] e^{-\beta_0^E [1 - e^{-\beta_1^E t_n}]}$$

The maximum likelihood equations are then

$$\frac{n}{1 - e^{-\beta_1^E(t_n)}} = \beta_0^E \quad (2.1)$$

$$\frac{n}{\beta_1^E} - \frac{n(t_n)}{e^{\beta_1^E(t_n)} - 1} - \sum_{i=1}^n t_i = 0 \quad (2.2)$$

It is important to note the β_0^E is a function of β_1^E , and finding the second parameter will allow us to simply solve equation 2.1 to obtain the first parameter. To find β_1 we use the previously described golden section routine to find the value of β_1 which results in equation 2.2 beginning as close to zero as possible.

2.3.3 The Logarithmic Model

From (Musa, Iannino, and Okumoto 1987) the likelihood equation for the logarithmic model is given as

$$L(\beta_1^L | n) = \frac{n! (\beta_1^L)^n \prod_{i=1}^n \frac{1}{1 + \beta_1^L t_i}}{[\ln(1 + \beta_1^L t_n)]^n}$$

From this, in (Farr 1996), Farr gives the maximum likelihood equations as

$$\beta_0^L = \frac{n}{\ln(1 + \beta_1^L t_n)} \quad (2.3)$$

$$\frac{1}{\beta_1^L} \sum_{i=1}^n \frac{1}{1 + \beta_1^L t_i} = \frac{nt_n}{(1 + \beta_1^L t_n) \ln(1 + \beta_1^L t_n)} \quad (2.4)$$

Again, β_0 is dependent on β_1 . The problem is to find an accurate estimate for β_1^L . We take the left hand side of equation 2.4 and subtract the right hand side. If we have chosen β_1^L correctly this equation should equal zero. We use the golden section routine to find the solution to this equation as described previously.

2.4 Comparison of Methods

To evaluate the effectiveness of the least squares method versus the maximum likelihood method we examine how each method affects the predictive ability of the exponential and logarithmic models on the data sets discussed in chapter 1. We use each method to estimate the parameters of each model as each new data point is added to the data set, and compute the MRE and SRE as discussed in chapter 1. We also keep track of how many times each method failed to obtain valid parameter estimates for a model.

Tables 2.1 through 2.4 present the results of our comparison of the two parameter estimation methods for the models we are studying, for both raw data and data which has been smoothed using two iterations of lump smoothing as recommended by Li in (Li and Malaiya 1996). The first four columns of each table present the MRE and SRE for the two methods side by side. Bold face entries in these columns denote the method which produced the most accurate results. The last two columns present the percentage of data points, which, when added to the analysis, caused that particular parameter estimation method to fail to find valid parameter estimates. Invalid parameter estimates are those where β_0 is negative, or β_1 exceeds the range of zero to one. In this case no contribution is made to the accuracy measures. So, it should be kept in mind that a high failure percentage and a low MRE or SRE does not necessarily indicate an accurate model and estimation technique. A high failure percentage does suggest that the data in question may not be appropriate to a given model or parameter estimation technique.

Examining the data for the exponential model we find that for every data set except T1 the mean relative error is lower for maximum likelihood estimation. However, this situation changes when smoothing is applied to the data. The overall best results for the long range predictability of the exponential model are obtained when the least squares method is used on smoothed data. Smoothing reduces the size of the data set, and so we would naturally expect some reduction in the predictive error, however, in this case the reduction seems to be more than can be accounted for by simply reducing the size of the data set. It should be noted that smoothing also appears to hamper the ability of the maximum likelihood method to find good parameter estimates. The SRE is always lowest for the case where maximum likelihood estimation is used on the raw data, suggesting that the best short term predictive accuracy comes when no smoothing is applied.

Examining the last two columns of table 2.1 we find that although the MLE method of parameter estimation can reliably find parameter estimates for the vast majority of cases, the least squares method often has difficulty doing so; in some cases failing to find accurate parameter estimates as often as 70% of the time when new data points are added to the data set. Smoothing causes the maximum likelihood method to fail to find valid parameter estimates more often and often does the same for the least squares method; although for some data sets smoothing actually makes it easier for the least squares method to find valid parameter estimates.

The results on the logarithmic model are more difficult to interpret. For many of the data sets examined, the best results were obtained when the least squares method was used on smoothed data. However, for a few data sets the best results were obtained when no smoothing was applied; for these data sets the least squares method occasionally produced better estimates than the maximum likelihood method, but sometimes did not. It is important to remember that smoothing reduces the size of the data set in question, and so we would expect to see some improvement whenever it is applied. In those cases where an improvement was seen when smoothing was applied it is difficult to say that it is because we have made a fundamental improvement in parameter estimation, and not because we have simply reduced the size of the data set. However, in some cases there does appear to be more improvement than can be accounted for by the reduction in the size of the data set. Smoothing also makes it more difficult for the maximum likelihood method to find good parameters. For some data sets it enabled the least squares method to find valid parameters a larger percentage of the time, but for many data sets smoothing made parameter estimation using least squares linear fitting more difficult. In all cases the best SRE was obtained using the maximum likelihood method on raw data.

Table 2.5 presents a summary of our findings. It gives the number of data sets for which each method resulted in the best long term predictive accuracy, for both the exponential and logarithmic models. Note that smoothing was always helpful to the accuracy of the exponential model, and usually helpful to the accuracy of the logarithmic model. Overall, least squares estimation applied to smoothed data appears to represent the best method for long term predictions; while short term prediction is best accomplished using the maximum likelihood method.

Table 2.1: Least Squares vs. Maximum Likelihood Estimation for the Exponential Model

Data	MRE		SRE		% Failure	
Set	LSQ	MLE	LSQ	MLE	LSQ	MLE
T1	0.792	2.163	1.287	0.041	5	1
T2	0.635	0.599	0.389	0.053	13	2
T3	1.358	1.117	7.040	0.068	5	3
T4	2.059	1.011	1.826	0.069	9	2
T5	1.846	0.498	2.262	0.007	22	0
T6	2.007	0.586	2.146	0.070	11	1
T14C	3.516	0.462	3.611	0.074	61	3
T17	1.395	0.449	1.026	0.074	55	3
T27	1.209	0.601	1.414	0.075	44	2
SS1A	1.531	0.488	1.537	0.033	72	1
SS1B	1.503	0.504	1.947	0.014	13	0
SS1C	1.610	0.500	1.723	0.017	29	0
SS2	1.059	0.506	2.285	0.022	77	1
SS3	5.352	0.504	6.627	0.017	13	0
SS4	1.766	0.494	2.493	0.022	67	1
CSR1	1.576	0.504	1.556	0.021	49	0
CSR2	2.324	0.485	2.528	0.030	40	1
CSR3	1.019	0.545	1.353	0.035	5	1

Table 2.2: LSQ vs. MLE for the Exponential Model, Smoothed Data

Data	MRE		SRE		% Failure	
Set	LSQ	MLE	LSQ	MLE	LSQ	MLE
T1	0.502	1.021	0.923	0.867	6	6
T2	0.345	0.803	0.367	0.884	17	17
T3	0.576	0.880	0.478	0.825	14	14
T4	1.497	0.759	2.758	0.863	14	14
T5	0.248	0.945	0.189	0.898	19	1
T6	0.506	0.909	1.320	0.909	14	14
T14C	0.220	0.876	0.227	0.875	25	25
T17	0.084	0.836	0.114	0.854	57	14
T27	0.416	0.756	0.286	0.852	33	17
SS1A	0.194	0.922	0.206	0.889	21	7
SS1B	0.331	0.929	0.342	0.898	3	3
SS1C	0.331	0.927	0.259	0.888	3	3
SS2	0.180	0.935	1.051	0.906	9	5
SS3	0.283	0.919	0.147	0.895	10	3
SS4	0.133	0.920	0.047	0.864	42	4
CSR1	0.398	0.924	0.177	0.880	39	2
CSR2	0.358	0.885	0.276	0.871	47	5
CSR3	0.238	0.851	0.128	0.859	14	7

Table 2.3: Least Squares vs. Maximum Likelihood Estimation for the Logarithmic Model

Data	MRE		SRE		% Failure	
Set	LSQ	MLE	LSQ	MLE	LSQ	MLE
T1	0.156	0.412	0.065	0.021	5	1
T2	0.161	0.393	0.099	0.044	7	2
T3	0.277	0.343	0.253	0.050	32	3
T4	0.496	0.364	0.151	0.045	15	2
T5	0.156	0.468	0.014	0.006	16	0
T6	0.249	0.414	0.089	0.036	16	1
T14C	0.401	0.433	0.142	0.073	69	3
T17	0.302	0.399	0.153	0.067	53	3
T27	0.558	0.405	0.152	0.061	49	2
SS1A	0.113	0.461	0.038	0.032	64	1
SS1B	0.278	0.474	0.024	0.011	2	0
SS1C	0.225	0.463	0.045	0.015	1	0
SS2	0.602	0.474	0.101	0.020	80	1
SS3	0.255	0.463	0.027	0.015	12	0
SS4	0.169	0.467	0.022	0.020	32	1
CSR1	0.834	0.377	0.108	0.011	64	0
CSR2	0.391	0.387	0.176	0.028	65	1
CSR3	0.244	0.403	0.075	0.026	11	1

Table 2.4: LSQ vs. MLE for the Logarithmic Model, Smoothed Data

Data	MRE		SRE		% Failure	
Set	LSQ	MLE	LSQ	MLE	LSQ	MLE
T1	0.302	0.925	0.282	0.875	6	6
T2	0.297	0.916	0.393	0.895	17	17
T3	0.416	0.865	0.396	0.848	29	14
T4	0.580	0.903	0.892	0.885	14	14
T5	0.151	0.944	0.047	0.898	19	1
T6	0.091	0.933	0.507	0.917	29	14
T14C	0.132	0.914	0.159	0.926	25	25
T17	0.103	0.873	0.110	0.862	57	14
T27	0.533	0.892	0.344	0.864	17	17
SS1A	0.146	0.925	0.137	0.889	21	7
SS1B	0.278	0.946	0.084	0.901	3	3
SS1C	0.245	0.940	0.150	0.891	3	3
SS2	0.210	0.936	0.425	0.906	45	5
SS3	0.187	0.939	0.070	0.904	10	3
SS4	0.118	0.930	0.047	0.870	38	4
CSR1	1.050	0.922	0.067	0.879	61	2
CSR2	0.035	0.904	0.066	0.869	58	5
CSR3	0.145	0.911	0.075	0.866	14	7

Table 2.5: Summary of Least Squares versus Maximum Likelihood

Method	Exp. Model	Log. Model
Raw LSQ	0	5
Raw MLE	0	3
Smth. LSQ	17	10
Smth. MLE	1	0

2.5 Stability of Estimates

When choosing a method of parameter estimation the stability of parameter estimates should also be considered. As new data points are added to a data set the resulting parameter estimates should not change drastically. Ideally, they should take on values which are very close to their final values early, otherwise long term predictive accuracy is likely to be low.

We have found that the maximum likelihood method provides parameter estimates which seem more stable than those produced by the least squares method. Stability of an estimated parameter is a difficult thing to measure, and one of the most useful tools for assessing the stability of parameters estimates is a simple plot of what the parameter estimates are after each new data point is taken into consideration. Figures 2.1 and 2.2 show such plots, for the exponential model applied to the T3 data set and the logarithmic model applied to the T4 data set respectively. These plots are typical of what is seen with other data sets, although each data set has its own characteristic profile of parameter stability these four plots give a good indication of how such plots look for the data examined in this study.

Examining these plots several things become apparent. First and foremost, estimates of β_0 produced by the maximum likelihood method are much more stable than those produced by the least squares method. Estimates produced by the least squares method tend to jump around as each new data point is added. Estimates for β_0 produced by the least squares method tend to rise as each new data point is added, at least over the long term. This observation holds for both the exponential and the logarithmic models. In contrast, estimates for β_0 produced by the maximum likelihood method rise much slower over time, although they still appear to grow slowly as each new data point is added. However, this growth is largely predictable, each new data point that is taken into consideration results in a small increase in the estimated parameter, and never the huge rise or fall that we see with the least squares method. In fact, it appears that in the great majority of case the maximum likelihood method produces estimates of β_0 which are the same as the number of data points taken into consideration. As discussed in chapter 3 the parameter β_0 for the exponential model can be interpreted as the total number of faults present in the system. A parameter estimation method which tells us that we have found all the faults in the system early on, and continues to do so as more faults are found, is to be suspect. The parameter β_0 for the logarithmic model has a somewhat different interpretation, and so the problem there is not so severe. However, it appears that this method might still produce better results when trying to make long range predictions.

The situation is somewhat different for the β_1 parameter. In general, it appears that the least squares method produces parameter estimates for β_1 which are more stable than those produced by the maximum likelihood method. The plots in this case can be misleading, however, because the estimates produced by the least squares method are very close to zero, several orders of magnitude lower than those produced by the maximum likelihood method. Consequently, when the results of both estimation methods are plotted together the least squares estimates can look smoother than they really are. Estimates produced by the least squares method can vary a great deal in terms of relative value between consecutive data points, but the absolute difference in terms of the numeric value of these estimates is less than the difference between consecutive estimates of the parameters obtain using the maximum likelihood method. In generally, however, estimates of β_1 appears to be more stable than those obtained for β_0 , regardless of the method used.

In order to obtain a better understanding of the stability of parameter estimates we would like to have a quantitative measure of stability. We recognize that regardless of the method used there will be outliers, points at which the estimates produced are either illegal or so far from the estimates made using one data point more or less that they should not be taken into consideration because in practice they would be recognized for what they are and thrown out. The method we use to measure parameter stability is the standard deviation of the parameter estimates, adjusted for outliers. We first compute the standard deviation for all valid estimates

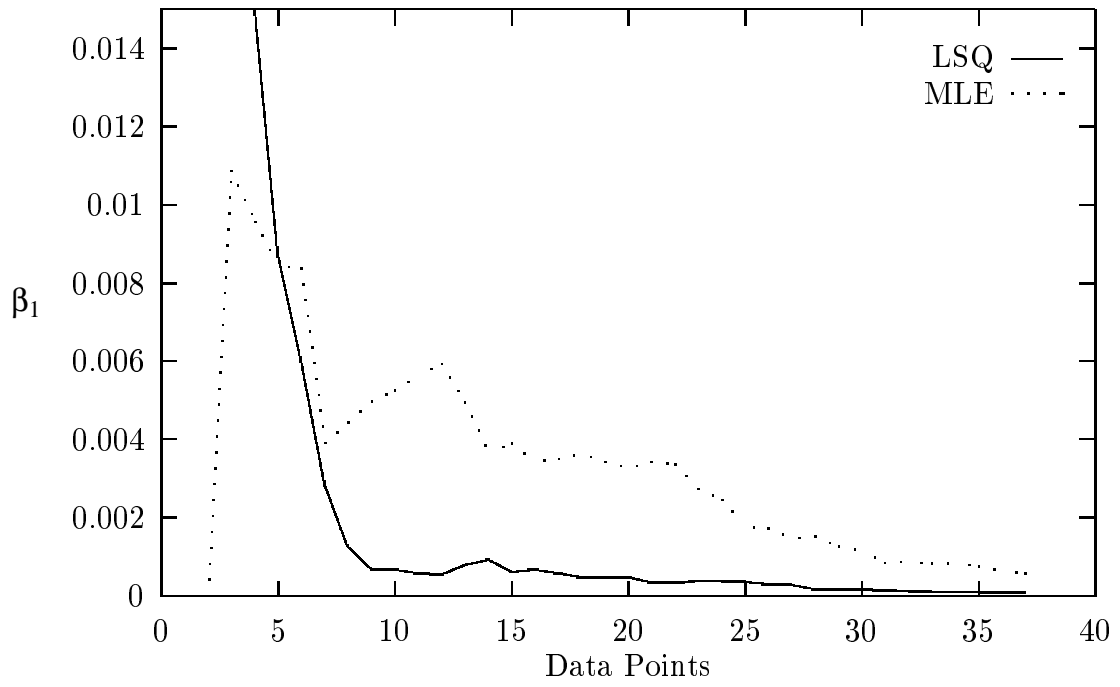
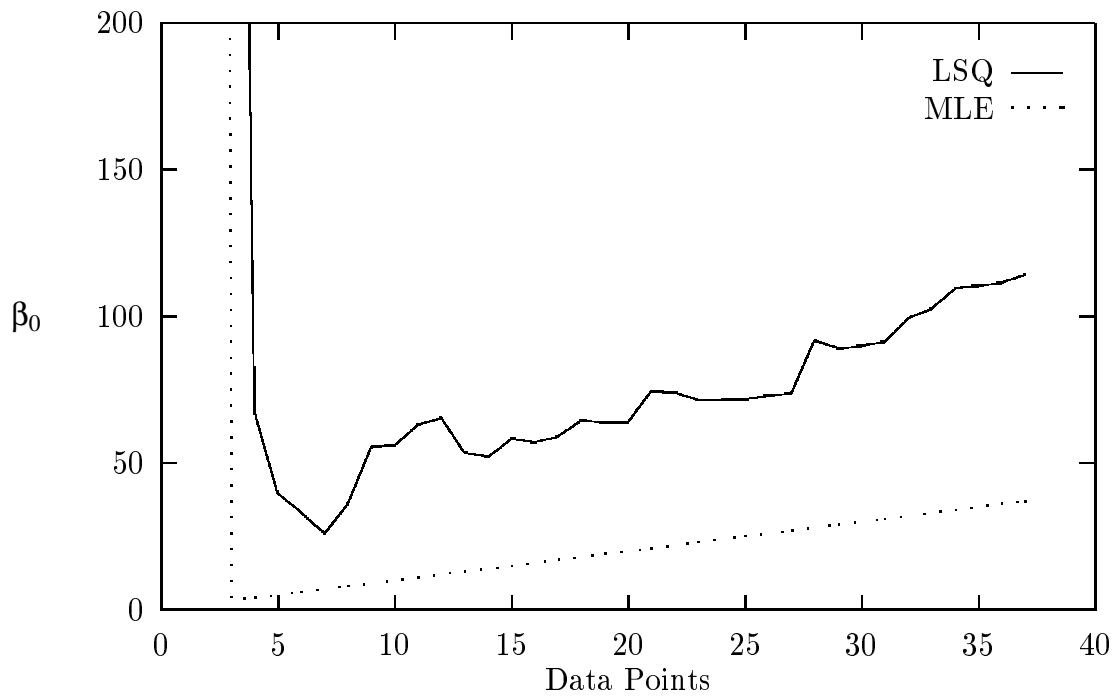


Figure 2.1: Parameter Stability for Exponential Model, Data Set T3

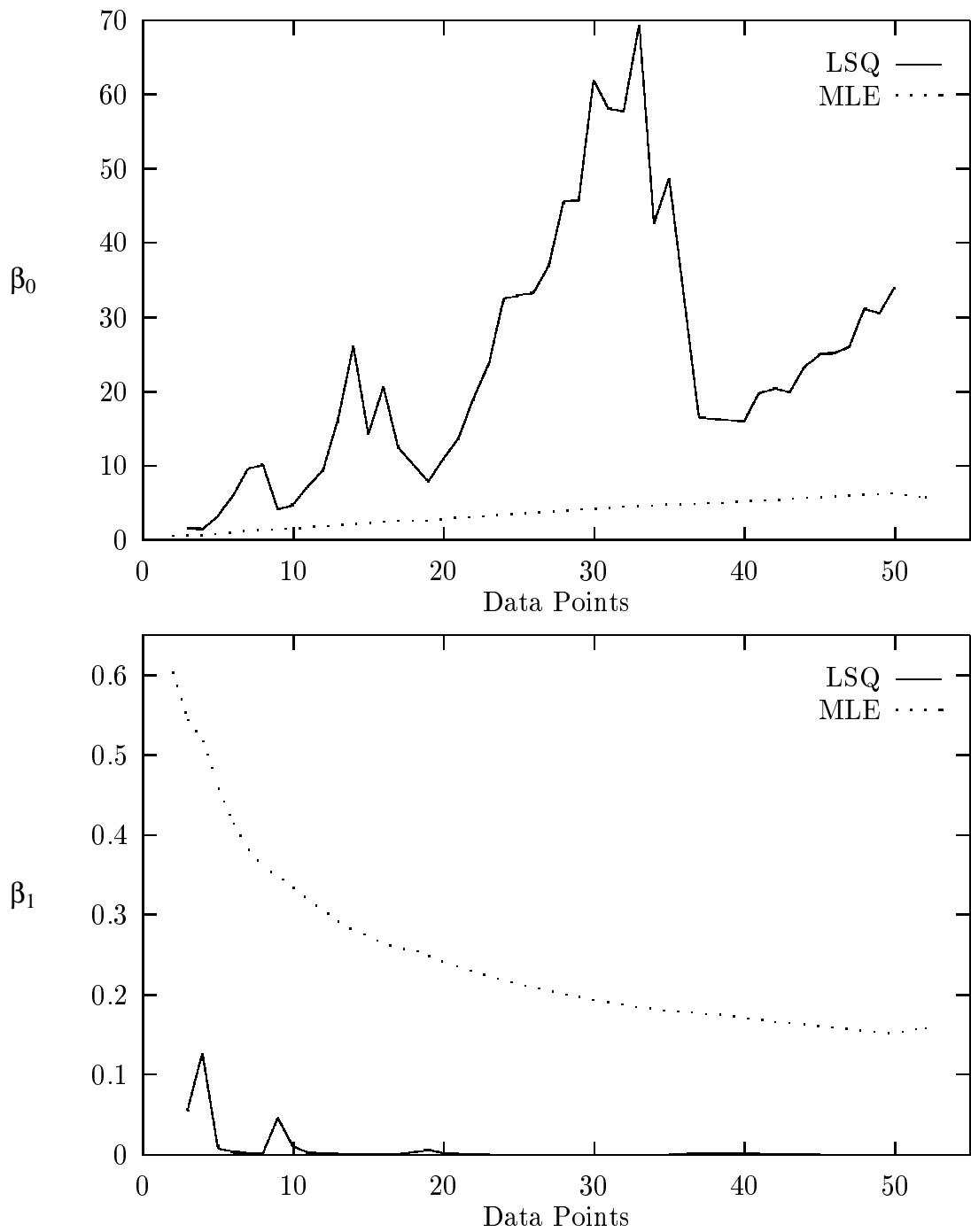


Figure 2.2: Parameter Stability for Logarithmic Model, Data Set T4

of a parameter obtained as new data points are added, and then throw out those data points which are more than three standard deviations from the average. We then recompute the standard deviation and take that as a quantitative measure of the stability of parameter estimates. When the standard deviation for a data set and parameter is high, then it can be assumed that the method in question is having difficulty obtaining stable estimates for that parameter on that data set. When it is low, we can assume that each new parameter estimate made differs very little from its predecessor.

Tables 2.6 and 2.7 give the standard deviation in parameter estimates for both the least squares and the maximum likelihood method for both the exponential and logarithmic model. Here we can confirm what our previous examination of the plots suggested. The least squares method usually but not always produces more stable estimates of β_1 for both models. The maximum likelihood method always produces more stable estimates for β_0 , which appears to be the more important parameter.

Table 2.6: Standard Deviation of Parameter β_0

Data Set	Exp. Model		Log. Model	
	LSQ	MLE	LSQ	MLE
T1	94.614	38.385	14.805	4.073
T2	50.389	14.701	5.852	1.627
T3	22.686	10.073	3.950	1.086
T4	83.045	14.412	14.778	1.741
T5	3902.965	239.022	492.926	21.344
T6	290.795	19.912	44.506	2.918
T14C	120.659	9.494	34.788	0.947
T17	62.251	10.073	19.202	1.254
T27	15.791	10.941	61.418	1.062
SS1A	1707.691	31.455	168.851	3.000
SS1B	3358.342	107.384	239.011	8.552
SS1C	1078.536	79.093	146.448	6.624
SS2	2687.199	54.554	224.316	4.642
SS3	2389.219	79.382	1312.875	6.710
SS4	1663.831	55.708	1061.265	4.933
CSR1	430.368	113.735	577.475	12.946
CSR2	116.159	36.364	28.382	4.239
CSR3	104.593	29.145	25.532	3.655

The least squares technique gives values of β_0 that can vary greatly. This is attributable to the fact that the estimates obtained using this technique start very high, and then generally climb as each new data point is added. Over a hundred or more data points the difference between estimates made when only a few data points were present and those made when nearly all data points are being used to fit the model can be very great. This problem is particularly acute with the exponential model, which also gives the maximum likelihood method trouble.

It is worth noting that in this section we have only presented results obtained from raw data. The goal of smoothing, as suggested by Li (Li 1996) is to remove noise from the data set, and thus it would be expected that smoothing would increase the stability of parameter estimates. We have repeated these experiments using data that has been through two iterations of lump smoothing as suggested by Li (Li 1996). It was found that in general our previous observations hold, smoothing does not allow the least square method to make estimates for β_0 which are better than those produced by the maximum likelihood method, and the most stable estimates for β_1 still come from the least squares method. Lump smoothing, as described in chapter 1, reduces the number of data points in a data set, and for many of the data sets examined here it resulted in less than ten data points that could be considered to be valid and not outliers. So, even though smoothing can be used to increase the stability of parameter estimates, it should only be so employed when the resulting data set has enough data points to be useful. It should be noted that when the least squares method is used on smoothed data the parameter estimates fall in between the estimates for raw data, which seem to be high, and the maximum likelihood estimates which seem to be low as previously discussed. This may account for the good predictive

Table 2.7: Standard Deviation of Parameter β_1

Data Set	Exp. Model		Log. Model	
	LSQ	MLE	LSQ	MLE
T1	0.00046539	0.00421186	0.00195469	0.10690499
T2	0.00018215	0.00093655	0.00279334	0.02235004
T3	0.00294732	0.00260099	0.08298871	0.02963759
T4	0.00578285	0.00182699	0.02166522	0.10602395
T5	0.00000034	0.00001436	0.00000083	0.00174400
T6	0.01477523	0.01084148	0.12138019	0.10949254
T14C	0.00000029	0.00000066	0.00001609	0.00092972
T17	0.00000390	0.00005719	0.00006396	0.00342638
T27	0.00000043	0.00001108	0.00002267	0.00306720
SS1A	0.00000060	0.00000129	0.00001482	0.00123435
SS1B	0.00000452	0.00000915	0.00001886	0.00416182
SS1C	0.00000292	0.00000781	0.00016511	0.00394094
SS2	0.00000211	0.00000086	0.00006130	0.00222910
SS3	0.00000107	0.00000480	0.00000532	0.00160457
SS4	0.00000097	0.00000042	0.00000666	0.00101723
CSR1	0.00036474	0.00669507	0.00725189	0.02064350
CSR2	0.00004885	0.00199305	0.02223113	0.00833303
CSR3	0.00257853	0.00886490	0.00830438	0.04823977

ability of this method, when the data is appropriate to its use.

2.6 Conclusion

We have outlined two different methods for estimating the parameters of the exponential and logarithmic software reliability growth models. We have presented details about this process which have previously been lacking in the literature. This will help future researchers implement and make use of these models.

Tables 2.1 through 2.4 demonstrate that the best short term predictions are obtained when the maximum likelihood method is used to estimate the models parameters on raw data. Based on table 2.5 it would be tempting to say that the best long term predictions are obtained when the least squares method is applied to smoothed data. Unfortunately, the least squares method has a great deal of difficulty arriving at valid parameter estimates on a number of data sets. Although the results are not as good, the maximum likelihood method does provide reasonable results when the data is smoothed, and very rarely has trouble finding valid parameter estimates. The marked difference between the ability of the two methods leads us to favor the maximum likelihood method; with the caveat that when the least squares method does not have difficulty finding parameter estimates it should be used instead.

Chapter 3

INTERPRETATION OF MODEL PARAMETERS

3.1 Introduction

A software reliability growth model (SRGM) can be regarded to be a mathematical expression which fits the experimental data. It may be obtained simply by observing the overall trend of reliability growth. However some of the models can be obtained analytically by making some assumptions about the software testing and debugging process. Some of these assumptions are simply to keep the analysis tractable. Others are more fundamental in nature and constitute modeling of the testing and debugging process itself.

An analytically obtained model has the advantage that its parameters have specific interpretations in terms of the testing process. An understanding of the underlying meaning of the parameters gives us a valuable insight into the process.

1. If we know how a parameter arises, we can estimate it even before testing begins. Such *a priori* values, when estimated using past experience, can be used to do preliminary planning and resource allocation before testing begins (Malaiya 1991).
2. Experience with use of SRGMs suggests that in the beginning of testing, the initial test data yields very unstable parameter values and sometimes the parameter values obtained can be illegal in terms of the model. In such a situation, values estimated using static information can serve as a check. Static values may also be used to stabilize the fitted dynamic values of models early in the test process, while accurate estimates are still difficult to come by. We address this in detail in chapter 4.
3. Parameters that have an interpretation characterize the testing and debugging process quantitatively. Their values can give us an insight into the process. They may help answer questions about how the inherent defect density can be reduced or how testing can be made more efficient.

This chapter examines the parameters of the exponential and the logarithmic models. A derivation of the exponential model is discussed and a new interpretation for the parameters of the logarithmic model is presented. The next section analytically presents the interpretations of the parameters of the two models. Section 3.4 discusses estimation of the parameters based on information about the development process.

We will chiefly be concerned with the logarithmic model in this chapter. As previously discussed, this model has been shown to be superior to other models in a wide variety of cases. It is also relatively easy to use, with parameters that are easily found using standard techniques. Unfortunately, a concrete interpretation of these parameters has been lacking. Because the exponential model can be considered an approximation of the logarithmic model, we will consider estimating its parameter based on the parameters of the exponential model; followed by an attempt at direct estimation of the parameters. Consequently, we present an interpretation of the parameters of the exponential model first. The work presented here comes largely from (Malaiya and Denton 1997) by Malaiya and Denton.

3.2 Derivation of the Exponential Model

Here we give a derivation of the exponential model that gives its relationship with the test process. This will allow us to interpret the meaning of the two parameters of this model. Let $N(t)$ be the expected number of defects present in the system at time t . Let T_s be the average time needed for a single execution, which is very small compared with the overall testing duration. Let k_s be the expected fraction of existing faults exposed during a single execution. Then

$$\frac{dN(t)}{dt} T_s = -k_s N(t) \quad (3.1)$$

It would be convenient to replace T_s with something which can be easily estimated. Let T_L be the *linear execution time* (Musa, Iannino, and Okumoto 1987) which is defined as the total time needed if each instruction in the program was executed once and only once. This is given by

$$T_L = \frac{I_s Q_x}{r}$$

where I_s is the number of source statements, Q_x is the number of object (machine level) instructions per source instructions and r is the object instruction execution rate of the computer being used.

Let us define a new parameter

$$K = k_s \frac{T_L}{T_s}$$

where the ratio $\frac{T_L}{T_s}$ will depend on the program structure. Using this, equation 3.1 can be rewritten as

$$\frac{dN(t)}{dt} = -\frac{K}{T_L} N(t) \quad (3.2)$$

The per-fault hazard rate as given in equation 3.2 is K/T_L . Thus K , termed *fault exposure ratio* (Musa, Iannino, and Okumoto 1987) directly controls the efficiency of the testing process. If we assume that K is time invariant, then the above equation has the following solution:

$$N(t) = N_0 e^{-\frac{K}{T_L} t}$$

where N_0 is the initial number of defects. This may be expressed in a more familiar form as follows:

$$N_0 - N(t) = N_0 (1 - e^{-\frac{K}{T_L} t})$$

The left side of this equation corresponds to $\mu(t)$ for the exponential model, as given by equation 1.1. Thus the parameters β_0 and β_1 have the following interpretations:

$$\beta_0^E = N_0 \quad (3.3)$$

$$\beta_1^E = \frac{K}{T_L} \quad (3.4)$$

An interpretation of the parameters for the exponential model is quite straightforward. As $t \rightarrow \infty$, according to equation 1.1, $\mu(t) \rightarrow \beta_0^E$. Musa states that during debugging only about 5% new faults are introduced. Thus β_0^E is slightly greater than the initial number of faults, and can be taken to represent the total number of faults that will be encountered. The parameter β_1^E is the time scale factor, or the per fault hazard rate, as given by equation 3.4.

Experimental data suggests that the fault exposure ratio K actually varies during testing (Malaiya, von Mayrhauser, and Srimani 1993). We will denote the constant equivalent as determined by the application of the exponential model by \hat{K} .

3.3 Implications of the Logarithmic model

The logarithmic model has been found to have very good predictive capability in many cases. However to derive it from basic considerations requires one to make some assumptions as done in (Malaiya, von Mayrhauser, and Srimani 1993), (Musa, Iannino, and Okumoto 1987), and (Musa and Okumoto 1984). We show below that if the logarithmic model describes the test process, the fault exposure ratio is variable. We assume that this variation depends on the test process phase (Li and Malaiya 1996).

We begin by rearranging the mean value function for the logarithmic model given by equation 1.3. We can write for $\mu(t)$,

$$e^{\frac{\mu(t)}{\beta_0^L}} = (1 + \beta_1^L t) \quad (3.5)$$

Also, from equation 1.4

$$\lambda(t) = \frac{\beta_0^L \beta_1^L}{1 + \beta_1^L t}$$

The number of defects found at time t is given by $\mu(t)$, with the defect density at time t given by $D(t)$. If we multiply the defect density at time t by the number of source lines I_s , then we obtain the remaining defects. If we assume that defects are removed as soon as they are found then we can write the number of defects found as $\mu(t) = N_0 - I_s D(t)$. Substituting this result in for $\mu(t)$ and using equation 3.5 to substitute for $(1 + \beta_1^L t)$ we obtain

$$\lambda(t) = \beta_0^L \beta_1^L e^{-\frac{\mu(t)}{\beta_0^L}} = \beta_0^L \beta_1^L e^{-\frac{N_0 - I_s D(t)}{\beta_0^L}} \quad (3.6)$$

From equation 3.2, we know that the fault exposure ratio is given by

$$K(t) = T_L \frac{\lambda(t)}{N(t)}$$

Using equation 3.6 to substitute for $\lambda(t)$, we get

$$\begin{aligned} K(t) &= \frac{T_L}{I_s D} \beta_0^L \beta_1^L e^{-\frac{N_0 - I_s D(t)}{\beta_0^L}} \\ &= \left(\frac{T_L}{I_s D} \beta_0^L \beta_1^L e^{-\frac{N_0}{\beta_0^L}} \right) e^{-\frac{I_s D(t)}{\beta_0^L}} \end{aligned}$$

Which we can rewrite as

$$K(D) = \frac{\alpha_0}{D} e^{\alpha_1 D} \quad (3.7)$$

Here we have expressed the fault exposure ratio K as a function of defect density D instead of time t . The parameters α_0 and α_1 are given by,

$$\alpha_0 = \frac{\beta_0^L \beta_1^L Q_x e^{-\frac{N_0}{\beta_0^L}}}{r} \quad (3.8)$$

$$\alpha_1 = \frac{I_s}{\beta_0^L} \quad (3.9)$$

An interpretation of the parameters for the exponential model is quite straightforward. As $t \rightarrow \infty$, according to equation 1.1, $\mu(t) \rightarrow \beta_0^E$. Musa states that during debugging only about 5% new faults are introduced. Thus β_0^E is slightly greater than the initial number of faults, and can be taken to represent the total number of faults that will be encountered. The parameter β_1^E is the time scale factor, or the per fault hazard rate, as given by equation 3.4.

We now ready to present an interpretation of the parameters of the logarithmic model.

3.3.1 Interpretation of the Logarithmic Model Parameters

From equation 3.8 we can write

$$\beta_0^L = \frac{I_s}{\alpha_1}$$

Substituting this in equation 3.8 and solving for β_1^L , we get

$$\beta_1^L = \frac{\alpha_0 r \alpha_1}{Q_r I_s} e^{\frac{N_0 \alpha_1}{I_s}}$$

Let us now determine the meaning of α_0 and α_1 , in terms of the test process. Fig. 3.1 gives the variation of the fault exposure ratio K in terms of defect density. Note that for the exponential model the fault exposure ratio does not vary, but that for the logarithmic model it starts very high, then initially drops before rising again with defect density.

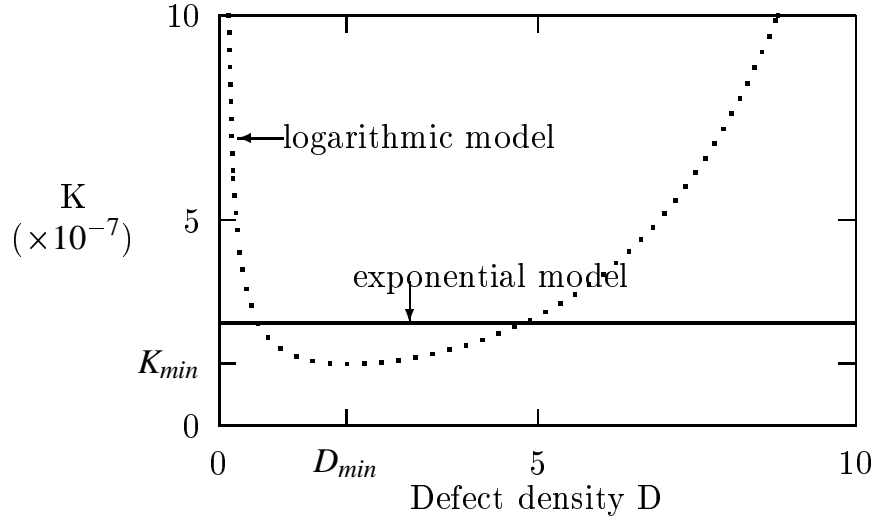


Figure 3.1: Variation of Fault Exposure Ratio with defect density

Let us denote by D_{min} the density at which K_{min} , the minimum value of K , occurs. Taking a derivative of K with respect to D using equation 3.7 and equating it to zero, we get

$$-\frac{\alpha_0}{D^2} e^{\alpha_1 D} + \frac{\alpha_0}{D} e^{\alpha_1 D} \alpha_1 = 0$$

which yields

$$D_{min} = \frac{1}{\alpha_1}$$

and the corresponding value of K is given by

$$K_{min} = \frac{\alpha_0 e}{D_{min}}$$

Thus both α_0 and α_1 depend on the test process,

$$\alpha_0 = \frac{K_{min} D_{min}}{e} \quad (3.10)$$

$$\alpha_1 = \frac{1}{D_{min}} \quad (3.11)$$

Using equations 3.8, 3.9, 3.9 and 3.11, we obtain this interpretation of the logarithmic model parameters.

$$\beta_0^L = I_s D_{min} \quad (3.12)$$

$$\beta_1^L = \frac{K_{min} r}{Q_x I_s} e^{\frac{D_0}{D_{min}}} \quad (3.13)$$

Here D_0 is the initial defect density. Equation 3.12 states that β_0^L is proportional to the software size and is controlled by how test effectiveness varies with defect density. The parameter β_1 depends on K_{min} , the minimum value of the fault exposure ratio. It is also dependent on the ratio $\frac{D_0}{D_{min}}$.

It should be noted that β_0^E and β_0^L , and β_1^E and β_1^L have the same dimensions. Table 3.1 below compares the interpretations of the parameters of the two models compared here.

Table 3.1: Comparison of model parameter interpretations

	Value Scale	Time Scale
Dimension	Defects	Per unit time
Exponential	$\beta_0^E \approx N_0 = D_0 I_s$	$\beta_1^E = \frac{K}{T_f}$
Logarithmic	$\beta_0^L = D_{min} I_s$	$\beta_1^L = \frac{K_{min}}{T_f} e^{\frac{D_0 - D_{min}}{D_{min}}}$

3.4 Estimation of Logarithmic Model Parameters

Estimating the parameter values for the logarithmic model is a significant challenge. We can take one of two possible approaches. In the first approach, we can first estimate the parameters of the exponential model and then compute β_0^L and β_1^L . In the second approach, we can calculate β_0^L and β_1^L from the interpretation introduced in section 3.3.1.

3.4.1 Estimation through β_0^E and β_1^E

The parameters of the exponential model β_0^E and β_1^E are easily interpreted and estimated. Total linear execution time can be computed directly based on processor speed and program size, and defect density can be estimated using a number of different techniques. According to Malaiya and Denton (Malaiya and Denton 1997) the fault exposure ratio can be estimated as $K = \frac{1.2 \times 10^{-6}}{D_0} e^{0.05 D_0}$. We observe that for a given data set, there is some relationship between β_0^E and β_0^L , and β_1^E and β_1^L (Malaiya 1991). This relationship can be used to estimate the parameters of the logarithmic model once the exponential model parameters have been estimated. To obtain this relationship, let us assume that both models project the same $\mu(t_f)$ where t_f is the end of the testing period. Let the number of defects remaining at time t_f be $\frac{N_0}{\alpha}$, $\alpha > 1$. For example, if testing finds and removes 90% of all the faults, then $\alpha = 10$. Then

$$\mu(t_f) = N_0 - \frac{N_0}{\alpha} = N_0 \left(1 - \frac{1}{\alpha}\right) \quad (3.14)$$

For the exponential model equation 3.14 will give,

$$\beta_0^E (1 - e^{-\beta_1^E t_f}) = N_0 \left(1 - \frac{1}{\alpha}\right)$$

since $N_0 \approx \beta_0^E$, we can rewrite this equation as

$$t_f = \frac{\ln(\alpha)}{\beta_1^E} \quad (3.15)$$

using the logarithmic model we can write equation 3.14 as

$$\beta_0^L \ln(1 + \beta_1^L t_f) = N_0 \left(1 - \frac{1}{\alpha}\right)$$

which can be rearranged as

$$t_f = \frac{1}{\beta_1^L} \left[e^{\frac{\beta_0^E}{\beta_0^L} \left(1 - \frac{1}{\alpha}\right)} - 1 \right] \quad (3.16)$$

Equating the right hand side of equations 3.15 and 3.16, and rearranging we get

$$\frac{\beta_0^E}{\beta_0^L} = \frac{1}{1 - \frac{1}{\alpha}} \ln \left[\frac{\beta_1^L}{\beta_1^E} \ln(\alpha) + 1 \right] \quad (3.17)$$

Let us now assume that in time t_f the failure intensity also declines by factor α . Thus according to the exponential model,

$$\beta_0^E \beta_1^E e^{-\beta_1^E t_f} = \frac{\beta_0^E \beta_1^E}{\alpha}$$

which can be solved to obtain

$$\beta_1^E = \frac{1}{t_f} \ln(\alpha) \quad (3.18)$$

Similarly the logarithmic model gives

$$\frac{\beta_0^L \beta_1^L}{1 + \beta_1^L t_f} = \frac{\beta_0^L \beta_1^L}{\alpha}$$

which can be written as

$$\beta_1^L = \frac{1}{t_f} (\alpha - 1) \quad (3.19)$$

From equations 3.18 and 3.19 we obtain

$$\frac{\beta_1^L}{\beta_1^E} = \frac{\alpha - 1}{\ln(\alpha)} \quad (3.20)$$

Using equation 3.20, we can rewrite equation 3.17 as

$$\frac{\beta_0^E}{\beta_0^L} = \frac{\ln(\alpha)}{1 - \frac{1}{\alpha}} \quad (3.21)$$

Thus, if we know α and the values for β_0^E and β_1^E , we can calculate β_1^L using equation 3.20 and β_0^L using equation 3.21.

Example 1: For a software system under test, the parameters β_0^E and β_1^E have been estimated to be 142 and 0.35×10^{-4} respectively. Testing will be continued until about 92% of all faults have been found. That gives

$$\alpha = \frac{100}{100 - 92} = 12.5$$

Equation 3.20 gives

$$\begin{aligned} \frac{\beta_1^L}{\beta_1^E} &= 4.55 \\ \beta_1^L &= 4.55 \times 0.35 \times 10^{-4} = 1.59 \times 10^{-4} \end{aligned}$$

and equation 3.17 gives

$$\begin{aligned} \frac{\beta_0^E}{\beta_0^L} &= 2.75 \\ \beta_0^L &= \frac{142}{2.75} = 51.6 \end{aligned}$$

3.4.2 Direct Estimation of β_0^L and β_1^L

An alternative to the above method is to use the interpretation of β_0^L and β_1^L in terms of D_{min} and K_{min} as given by equations 3.12 and 3.13. A reasonable estimate for K_{min} is 1.5×10^{-7} as suggested by the data given by Musa et al. (Musa, Iannino, and Okumoto 1987) (their table 5.6). Estimation of D_{min} , the defect density at which the minimum value of K occurs is more difficult. First the curve for K , as shown in figure 3.1 has a very flat minimum. That can make exact determination of D_{min} hard in the presence of normal statistical fluctuations. Secondly, the variation in K depends on the testing strategy used.

Available data sets suggest the following.

1. If the initial defect density D_0 is less than ten defects per thousand lines of source code (KLOC), the value of D_{min} is in the neighborhood of 2 defects/KLOC.
2. However if D_0 is higher, the resulting value of D_{min} is also higher. in many cases, taking $D_{min} = D_0/3$ yields a suitable first estimate.

Example 2: For the T2 data set, the initial defect density is 8.23 defects/KLOC and the size is approximately 6.92 KLOC (27.7K object lines). We do not know the instruction execution rate, however we can obtain the value of T_L using available information. Since Musa et al. have given the value of \hat{K} as 2.15×10^{-7} and the value of β_1^E can be calculated to be 1.42×10^{-5} , the value of T_L is $2.15 \times 10^{-7}/1.42 \times 10^{-5} = 1.51 \times 10^{-2}$. We will estimate the values of the logarithmic model parameters assuming $D_{min} = 2$ and $K_{min} = 1.5 \times 10^{-7}$.

From equations 3.12 and 3.13 we have these estimates,

$$\beta_0^L = I_s D_{min} = 6.92 \times 2 = 13.84$$

and

$$\begin{aligned} \beta_1^L &= \frac{K_{min}}{e} \frac{r}{Q_x I_s} e^{\frac{D_0}{D_{min}}} \\ &= \frac{1.5 \times 10^{-7}}{2.72} \frac{1}{1.5 \times 10^{-2}} e^{\frac{8.23}{2}} \\ &= 2.24 \times 10^{-4} \end{aligned}$$

Fitting of actual test data yields $\beta_0^L = 17.26$ and $\beta_1^L = 2.01 \times 10^{-4}$. Considering the fact that the few early points in the test data often yield values that can be easily off by an order of magnitude or can be illegal (negative), these estimates are quite good. The *a priori* estimates of these models can be better than the values obtained in the early phases of testing, but can not be expected to be as accurate as the final values obtained using actual test data.

3.5 Concluding Remarks

A new interpretation for the parameters of the logarithmic model has been proposed and we have shown how it can be used to estimate the value of these parameters. An alternative approach is to first estimate the parameters for the exponential model and then use them to estimate the logarithmic model parameters. In following sections we will address which method of estimating the parameters of the logarithmic model is more accurate, and how static estimates can be used to improve fitted parameters values.

Chapter 4

STABILIZATION

4.1 Introduction

Estimates of the time and effort required to bring a project to completion are most valuable at the beginning of a project, when schedules can be adjusted, additional personal hired, and other management decisions made. Unfortunately, this is also the period of time in which software reliability growth models are the most unreliable. At the beginning of testing failure data is very noisy, and a single new data point can result in a large change in model parameter estimates. This instability makes accurate, or even consistent, reliability estimates difficult. As an alternative, static estimates of reliability model parameters can be used. When an interpretation of model parameters is available and stated in terms of the development process, it becomes possible to make parameter estimates based on quantitative measurements of the development process and the code under test. We term such estimates static parameters because they are not effected by erroneous test failure data at the beginning of testing, but can not be expected to be more accurate than values fitted as testing progresses because they are based on the practitioners perceptions of the development, rather than the resulting product.

This leads us to suggest that some combination of static estimates and dynamically fitted values might allow for more accurate estimates early in the testing processes, without sacrificing the accuracy of dynamic estimates later on. We term such a scheme stabilization. Li originally proposed doing stabilization using a weighted average of static and dynamic parameters, but said nothing about how to set the weights (Li and Malaiya 1995). An alternative approach would be to use static parameters under certain criteria, and then replace them with fitted values later on. We examine both approaches here.

4.2 Estimation of Static Parameters

In order to examine a combination of static and dynamic parameters, we must first obtain static estimates for the data with which we will be working. Doing so requires that we know something about the development process itself, and so we are restricted to that subset of the previously examined data for which the appropriate information is available. This subset includes the first six data sets of Musa's T series, and the last two of the same series.

In order to estimate the parameters of the exponential model, we need three pieces of information. First, we need to know the initial defect density, which gives us β_0^E . To obtain β_1^E we need to know the fault exposure ratio K and the total linear execution time T_L . For the data in question we assume that there are 5% more faults present than the number of faults eventually detected, as suggested by Musa (Musa, Iannino, and Okumoto 1987). The processor speed of the systems under test is unknown, but we assume it to be 1 MIPS based on the time at which the data was collected. Since we know the number of object instructions in each program we simply divide this by one million to get the number of seconds required to execute each instruction in the program once. For each of the data sets examined Musa provides the fault exposure ratio. This information can be found in chapter 1, in table 1.1.

Estimation of the parameters of the logarithmic model is more difficult. As outlined in chapter 3, we have two options. We can use our estimates of the parameters of the exponential model to obtain the parameters

of the logarithmic model using equations 4.1 and 4.2 where α is defined as the number of defects originally present divided by the number of defects present at the end of testing. We assume that each defect found was corrected, combined with our previous assumption that 5% more defects exist than were found, we find that α equals 21. Alternatively we can use equations 4.3 and 4.4 to directly estimate β_0^L and β_1^L . We will estimate D_{min} as one third of the our assumed defect density when that defect density exceeds ten defects per thousand lines of code. Otherwise, we take D_{min} to be two, as we suggest in chapter 3. We assume K_{min} to be 1.5×10^{-7} , also as suggest in chapter 3.

$$\beta_0^L = \frac{\beta_0^E \times 1 - \frac{1}{\alpha}}{\ln(\alpha)} \quad (4.1)$$

$$\beta_1^L = \frac{\alpha - 1}{\ln(\alpha)} \beta_1^E \quad (4.2)$$

$$\beta_0^L = I_s D_{min} \quad (4.3)$$

$$\beta_1^L = \frac{K_{min}}{T_L e} e^{D_{min}} \quad (4.4)$$

It should be noted that here we are working with known values for the initial defect density, fault exposure ratio, and linear execution time. The linear execution time of a program should always be available during testing; various tools can provide a count of the machine instructions and processor speed is readily available. The defect density will not be known in practice, but may be estimated in a number of ways. Models such as the Rome lab model (Lab 1992) and the Malaiya et al static defect density model (Malaiya and Denton 1997) allow for defect density to be estimated prior to the start of testing, and evidence suggests that defect density of a program often matches that of similar software previously developed by the same organization (Musa, Iannino, and Okumoto 1987). In chapter 5 we present a method for estimating the total number of defects, and thus defect density, based on coverage. We feel justified in using the known value here even though in practice it will have to be estimated; perhaps not as accurately as would like but hopefully close enough for our purposes.

The one value which is hard to estimate in practice is the overall fault exposure ratio. The values presented by Musa are exact, calculated for each data set at the end of testing. If these values are used to make our estimate then the static parameters of the exponential model, and by extension any estimates of the logarithmic model parameters made with them, will be very close to the true values. As a result, the long term predictive accuracy of these models will be better than what we would observe if we where re-fitting the model parameters after each failure; because we would be able to skip the early, noisy, portion of the data and obtain true values immediately. Thus, any results we might obtain performing stabilization under these conditions would be uncharacteristically good and not accurately reflect the value of the technique in practice. Nonetheless, we do our initial round of experiments using these known values of K . We do this to demonstrate that stabilization can be of practical value, provided that model parameters can be accurately estimated at the beginning of testing. Note that this is not an issue when we perform stabilization on the logarithmic model using the direct method of parameter estimation, because in this case we do not need to estimate K .

Table 4.1 gives the values for the static parameters that we will use in our experiments, estimated as outlined above.

4.3 Stabilization Techniques

We will examine two different stabilization techniques in this study, with several variations for each. The first method we will term the replacement method. The idea here is that static estimates provide an approximation of the model parameters, and that we can expect dynamic estimates to be no less than one-half and no more than twice this approximate value. When the dynamic parameters do exceed this range it is an indication that the most recently added data points have caused the fitting routines to fail and produce erroneous parameter estimates. In this case these dynamic estimates should be thrown out, and the static parameters used in their place. We consider four variations on this scheme here. We consider the case where β_0 is replaced when the dynamic estimates for this parameter exceed the acceptable range, and β_1 is never replaced. We also consider

Table 4.1: Static Parameters Used by Data Set, Known K

Data Set	Exponential Model		Logarithmic Direct Method		Logarithmic Indirect Method	
	β_0	β_1	β_0	β_1	β_0	β_1
T1	142.800	0.00000862	47.600	0.00005108	47.649	0.00005661
T2	56.700	0.00000776	33.240	0.00001097	18.920	0.00005099
T3	39.900	0.00001756	28.080	0.00000977	13.314	0.00011538
T4	55.650	0.00003164	40.200	0.00000658	18.569	0.00020786
T5	872.550	0.00000017	2934.000	0.00000003	291.151	0.00000113
T6	76.650	0.00006965	25.550	0.00019445	25.576	0.00045754
T17	39.900	0.00000733	30.950	0.00000324	13.314	0.00004818
T27	43.050	0.00000240	63.050	0.00000087	14.365	0.00001578

the converse, where β_1 is subject to replacement and β_0 is not. We then consider the cases where both parameters are subject to replacement separate from each other, and the case where if one parameter exceeds the acceptable range both parameters are replaced.

An alternative method would be to assign weights to the static and dynamic estimates and then take a weighted average of these two estimates. This approach was first proposed by Li and Malaiya in (Li and Malaiya 1995). We term the weight on the static parameters k , where k must be less than or equal to one. If we denote static parameter estimates with an S , and dynamically fitted values with a D , then the estimates used to make model predictions can be found to be

$$\begin{aligned}\beta_0 &= k\beta_0^S + (1-k)\beta_0^D \\ \beta_1 &= k\beta_1^S + (1-k)\beta_1^D\end{aligned}$$

Here we examine nine different methods for determining k . Several of these methods will make the value of k a function of the trend present in the collected data as measured by the Laplace factor. As a trend becomes more apparent, k decreases. This has the effect of making dynamic parameters more important as the data becomes more stable.

The Laplace factor is an analytic measure of trend which can be computed after any given failure i (Kanoun and Laprie 1996). This is important, because it means that we can evaluate the amount of trend present after each data point is added. Equation 4.5 gives the formula for the Laplace factor.

$$u(i) = \frac{\frac{1}{i-1}(\sum_{n=1}^{i-1} \sum_{j=1}^n \theta_j) - \frac{\sum_{j=1}^i \theta_j}{2}}{\sum_{j=1}^i \theta_j \sqrt{\frac{1}{12(i-1)}}} \quad (4.5)$$

where θ_j is the time between the j^{th} and $j-1^{\text{th}}$ failures.

Negative values of the Laplace factor indicate a decrease in failure intensity and thus a rise in software reliability. Positive values indicate that failure intensity is rising. Values above or below two indicate that a stable trend has developed in the data. At a significance level of 5%, there is a trend if $|u(i)| > 1.96$. Values of the Laplace factor outside the ± 1.96 range indicate that the data conforms to a homogeneous Poisson process, and that a reliable trend has developed.

For purposes of stabilization we are concerned with how close a data set is to exceeding the critical value of 1.96 and showing a stable trend. For this reason we divide the Laplace factor by 1.96 to obtain a percentage of how close the data is to having a trend. Values greater than 100% are clamped at 100%. Because stabilization is only concerned with parameter estimates, and not the direction of trend, we take the absolute value of the results. This gives us what we term the normalized Laplace factor, L where

$$L = \left| \frac{u(i)}{1.96} \right|$$

We examine nine different methods for determining k , as described below.

1. *Constant k* It may be that some constant value of k can provide for accurate reliability projections. Here we examine the case where k equals 0.25, 0.50 and 0.75.
2. *Linear k* We investigate the case where the value of k equals one minus the normalized Laplace factor up until L exceeds one. This allows the value of k to depend linearly on the amount of trend in the data until a stable trend develops, at which time we discontinue using static estimates and depend completely on dynamic parameters.
3. *Non-linear k* It may well be that the relationship between L and k is non-linear. We examine this by investigating the cases where $k = 1 - L^2$ and $k = e^{-L}$.
4. *Step Functions* We examine two cases where k is a step function of L . In each case the steps are all of equal size, with the first step having k equal one with later steps decreasing in consistent increments until $k = 0$ when L equals 1. The first step function examined has three steps, the second has five.
5. *Static Until Stable* It is possible that once a stable trend has developed there is no reason to use static estimates. It may also be that dynamic estimates are of no use before a stable trend develops. We examine the case where $k = 1$ for $L < 1$, and $k = 0$ for $L \geq 1$.

Figure 4.1 shows a plot of the weight on the static parameters, as determined by the linear and non-linear functions of the Laplace factor. Figure 4.2 shows the same thing, this time for the three and five step functions considered.

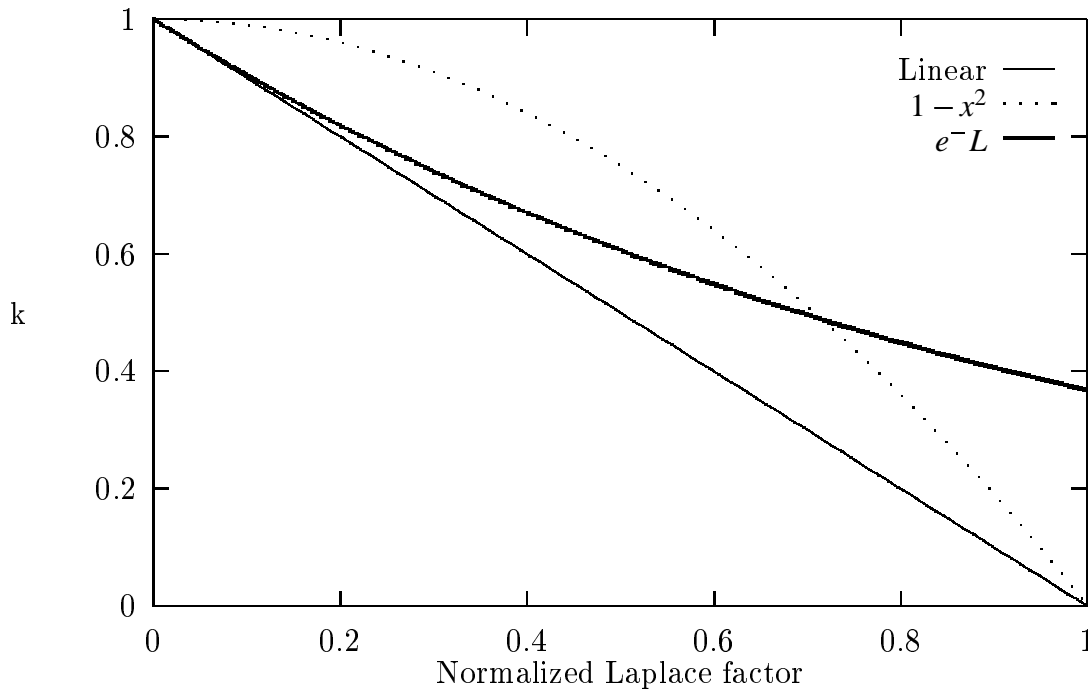


Figure 4.1: Linear and Non-Linear Stabilization Functions

4.4 Experiments

Because our interest is in how stabilization effects the long and short term predictive ability of software reliability growth models, we examine the mean relative error and the mean predictive error when stabilization is applied compared to when it is not. The goal of stabilization is to deal with noisy or erroneous failure data at the beginning of testing, which is similar to the goal of data smoothing which is too filter out such data points

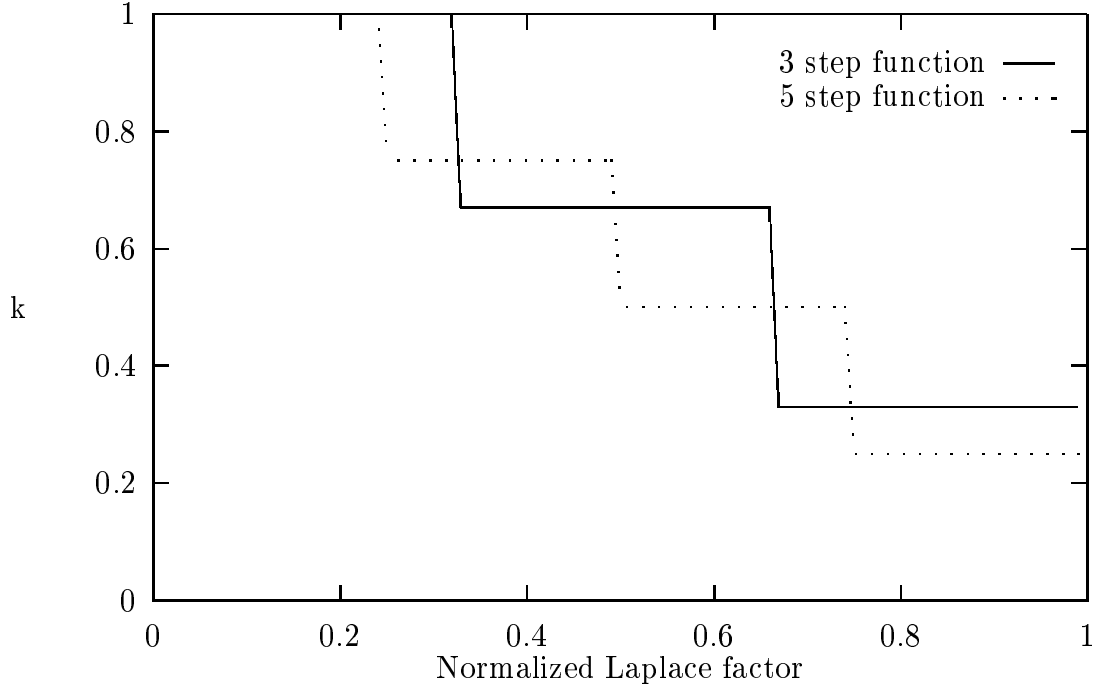


Figure 4.2: Stabilization Step Functions

across the entire data set. We examine how these two methods work together, applying stabilization to both raw data and data which has been smoothed using two iterations of lump smoothing as recommend by Li (Li and Malaiya 1996).

In all of our experiments we make use of the maximum likelihood method to estimate the dynamically fitted parameter values. Although our results in chapter 2 suggest that the least squares method result in better long term accuracy, that method is also highly subject to producing illegal parameters; so we choose to go with the maximum likelihood method. Stabilization is essentially a meta-estimation method, obtaining the final values for the parameters after examining the fitted parameters, the Laplace factor, the data, and other factors to determine how the static and dynamic estimates will be used to find the final parameter estimates. This makes it easy to evaluate our long and short term prediction measures. As each data point is added to the data set we re-fit the dynamic parameters and recalculate the Laplace factor for the data accumulated up to that point. We use the static parameters to replace or average with the fitted parameters as required, and then observe the predictions made by the model.

4.4.1 Results From Initial Experiments

Tables 4.2 through 4.4 present the results for the replacement method, comparing dynamic estimates using no stabilization against the four different variations on the replacement method. The rows marked “Replacement of β_0 ” and “Replacement of β_1 ” give the results for the cases where only the appropriate parameter is subject to replacement. The rows marked “Independent” give the results where each parameter is subject to replacement independent from the other, and the rows marked Dual Replacement present the results for the case where either parameter falling outside the acceptable range causes both to be replaced.

Table 4.5 presents the results of using the nine different weighted average stabilization techniques on the exponential model, for both raw and smoothed data. Tables 4.6 and 4.7 present the same results for the logarithmic model, using both the direct and indirect methods of static parameter estimation respectively.

Table 4.2: Results of Parameter Replacement Method for Exponential Model

	T1	T2	T3	T4	T5	T6	T17	T27
MRE for Raw Data								
No Stabilization	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
Replacement of β_0	0.274	0.151	0.163	0.168	0.140	0.156	0.172	0.168
Replacement of β_1	0.784	0.957	2.152	1.914	0.520	0.999	1.341	1.776
Independent	0.512	0.480	0.374	0.266	0.134	0.728	0.260	0.141
Dual Replacement	0.439	0.402	0.272	0.150	0.022	0.687	0.139	0.050
MRE for Smooth Data								
No Stabilization	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
Replacement of β_0	0.095	0.237	0.202	0.205	0.061	0.207	0.207	0.237
Replacement of β_1	1.280	6.294	7.277	8.695	0.971	4.379	15.309	14.274
Independent	0.439	0.402	0.272	0.150	0.022	0.687	0.139	0.049
Dual Replacement	0.439	0.402	0.272	0.150	0.022	0.687	0.139	0.050
	T1	T2	T3	T4	T5	T6	T17	T27
SRE for Raw Data								
No Stabilization	0.041	0.053	0.068	0.069	0.007	0.070	0.074	0.075
Replacement of β_0	2.713	1.835	1.522	1.819	4.535	2.112	1.493	1.590
Replacement of β_1	0.829	0.815	0.843	0.988	0.362	0.942	1.814	3.222
Independent	0.759	0.735	0.746	0.674	0.493	0.805	0.344	0.574
Dual Replacement	0.709	0.676	0.690	0.616	0.587	0.770	0.242	0.715
SRE for Smooth Data								
No Stabilization	0.867	0.884	0.825	0.863	0.898	0.909	0.854	0.852
Replacement of β_0	1.402	0.548	0.407	0.494	2.635	0.567	0.387	0.526
Replacement of β_1	0.908	3.104	2.580	5.313	1.042	3.772	17.162	31.448
Independent	0.658	0.566	0.482	0.421	0.565	0.720	0.185	0.478
Dual Replacement	0.658	0.566	0.482	0.421	0.565	0.720	0.185	0.478

Table 4.3: Results of Parameter Replacement Method for Logarithmic Model, Direct Estimation

	T1	T2	T3	T4	T5	T6	T17	T27
MRE for Raw Data								
No Stabilization	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
Replacement of β_0	2.382	2.706	3.457	3.745	25.717	1.516	3.645	9.999
Replacement of β_1	0.894	0.909	0.938	0.963	0.973	0.943	0.937	0.897
Independent	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
Dual Replacement	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
MRE for Smooth Data								
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Replacement of β_0	2.742	2.888	3.524	4.216	26.139	1.831	3.643	10.311
Replacement of β_1	0.988	0.988	0.987	0.995	0.997	0.994	0.987	0.982
Independent	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
Dual Replacement	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
	T1	T2	T3	T4	T5	T6	T17	T27
SRE for Raw Data								
No Stabilization	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
Replacement of β_0	8.126	7.184	7.196	9.365	118.010	5.207	9.510	24.496
Replacement of β_1	0.918	0.935	0.969	0.986	0.974	0.951	0.954	0.919
Independent	0.594	0.658	0.832	0.913	0.269	0.807	0.638	0.460
Dual Replacement	0.594	0.658	0.832	0.913	0.269	0.807	0.638	0.460
SRE for Smooth Data								
No Stabilization	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
Replacement of β_0	7.465	4.638	4.774	6.344	90.319	3.303	4.978	16.050
Replacement of β_1	0.990	0.991	0.991	0.997	0.997	0.995	0.989	0.985
Independent	0.549	0.564	0.713	0.844	0.266	0.772	0.584	0.624
Dual Replacement	0.549	0.564	0.713	0.844	0.266	0.772	0.584	0.624

Table 4.4: Results of Parameter Replacement Method for Logarithmic Model, Indirect Estimation

	T1	T2	T3	T4	T5	T6	T17	T27
MRE for Raw Data								
No Stabilization	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
Replacement of β_0	2.386	2.164	2.170	2.288	2.977	1.519	1.997	2.759
Replacement of β_1	0.889	0.872	0.841	0.827	0.847	0.900	0.822	0.763
Independent	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
Dual Replacement	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
MRE for Smooth Data								
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Replacement of β_0	2.746	2.320	2.217	2.614	3.040	1.834	1.996	2.865
Replacement of β_1	0.987	0.983	0.968	0.977	0.984	0.990	0.963	0.958
Independent	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
Dual Replacement	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
	T1	T2	T3	T4	T5	T6	T17	T27
SRE for Raw Data								
No Stabilization	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
Replacement of β_0	8.135	5.987	4.829	6.182	16.715	5.214	5.782	7.713
Replacement of β_1	0.912	0.898	0.896	0.884	0.791	0.906	0.820	0.728
Independent	0.563	0.527	0.567	0.435	0.915	0.616	0.212	1.014
Dual Replacement	0.563	0.527	0.567	0.435	0.915	0.616	0.212	1.014
SRE for Smooth Data								
No Stabilization	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
Replacement of β_0	7.474	3.813	3.107	4.089	12.593	3.307	2.857	4.827
Replacement of β_1	0.990	0.986	0.975	0.982	0.979	0.991	0.965	0.957
Independent	0.518	0.427	0.381	0.290	0.846	0.569	0.111	0.733
Dual Replacement	0.518	0.427	0.381	0.290	0.846	0.569	0.111	0.733

Table 4.5: Weighted Average Method For The Exponential Model

MRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
$K = 0.25$	0.798	0.465	1.040	0.570	0.365	0.360	0.586	1.240
$K = 0.50$	0.299	0.310	0.793	0.630	0.229	0.243	0.489	0.861
$K = 0.75$	0.094	0.140	0.422	0.378	0.095	0.123	0.264	0.413
Linear K	0.802	0.317	1.014	0.454	0.423	0.328	0.397	0.547
$K = 1 - L^2$	0.502	0.252	0.969	0.586	0.400	0.290	0.195	0.247
$K = e^{-L}$	0.500	0.263	0.899	0.634	0.409	0.299	0.325	0.473
3 Step Function	0.708	0.298	0.966	0.571	0.404	0.402	0.182	0.202
5 Step Function	0.434	0.322	0.272	0.197	0.373	0.610	0.153	0.121
Static Until Stable	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
Static Model	0.439	0.402	0.272	0.150	0.022	0.687	0.139	0.050
MRE for Smooth Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
$K = 0.25$	0.794	1.843	2.365	3.008	0.701	1.220	5.097	9.927
$K = 0.50$	0.532	1.940	2.415	3.077	0.456	1.244	5.406	7.083
$K = 0.75$	0.242	1.246	1.514	1.881	0.205	0.829	3.356	3.545
Linear K	0.996	1.627	2.414	2.092	0.633	0.776	5.436	9.987
$K = 1 - L^2$	0.970	2.043	2.724	2.784	0.516	0.904	3.568	8.115
$K = e^{-L}$	0.940	2.212	2.687	3.292	0.518	1.189	4.791	7.619
3 Step Function	0.947	2.098	2.607	3.207	0.518	1.085	4.191	9.264
5 Step Function	0.826	0.705	0.569	0.392	0.260	0.687	0.139	0.376
Static Until Stable	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
Static Model	0.439	0.402	0.272	0.150	0.022	0.687	0.139	0.050
SRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.041	0.053	0.068	0.069	0.007	0.070	0.074	0.075
$K = 0.25$	0.702	0.456	0.377	0.490	1.178	0.534	0.597	0.969
$K = 0.50$	1.398	0.929	0.778	0.993	2.325	1.090	1.051	1.566
$K = 0.75$	1.945	1.263	1.039	1.325	3.319	1.475	1.207	1.666
Linear K	0.738	0.751	0.556	0.714	1.369	0.723	0.768	1.300
$K = 1 - L^2$	0.594	0.646	0.521	0.780	1.393	0.740	0.720	0.895
$K = e^{-L}$	0.901	0.899	0.680	0.906	1.676	0.928	0.939	1.370
3 Step Function	0.397	0.582	0.597	0.745	0.980	0.818	0.701	0.782
5 Step Function	0.130	0.407	0.446	0.422	0.138	0.534	0.218	0.589
Static Until Stable	0.041	0.053	0.068	0.069	0.007	0.070	0.074	0.075
Static Model	0.709	0.676	0.690	0.616	0.587	0.770	0.242	0.715
SRE for Smooth Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.867	0.884	0.825	0.863	0.898	0.909	0.854	0.852
$K = 0.25$	0.556	0.718	0.692	1.242	0.676	0.897	3.935	8.279
$K = 0.50$	0.604	0.647	0.589	1.316	1.079	0.872	4.727	9.678
$K = 0.75$	0.716	0.419	0.327	0.874	1.557	0.604	3.306	6.639
Linear K	0.920	0.624	0.788	0.780	1.288	0.506	4.726	8.302
$K = 1 - L^2$	0.995	0.855	0.847	1.009	0.996	0.632	3.549	10.126
$K = e^{-L}$	1.054	0.914	0.876	1.346	1.270	0.857	4.408	9.970
3 Step Function	1.067	0.862	0.815	1.248	0.837	0.785	4.063	9.500
5 Step Function	0.890	0.795	0.806	0.569	0.648	0.720	0.185	0.684
Static Until Stable	0.867	0.884	0.825	0.863	0.898	0.909	0.854	0.852
Static Model	0.658	0.566	0.482	0.421	0.565	0.720	0.185	0.478

Table 4.6: Weighted Average Method For Logarithmic Model, Direct Estimation

MRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
$K = 0.25$	0.251	0.338	0.556	0.618	5.899	0.143	0.558	2.110
$K = 0.50$	0.842	0.991	1.361	1.495	11.793	0.401	1.411	4.422
$K = 0.75$	1.299	1.481	1.970	2.170	16.707	0.642	2.046	6.313
Linear K	0.431	0.450	0.568	0.926	2.218	0.416	1.097	2.542
$K = 1 - L^2$	0.447	0.553	0.657	1.154	2.522	0.455	1.344	2.973
$K = e^{-L}$	0.451	0.624	0.803	1.138	2.580	0.474	1.380	3.066
3 Step Function	0.365	0.437	0.624	0.908	1.458	0.508	1.142	1.845
5 Step Function	0.358	0.310	0.374	0.494	0.389	0.636	0.450	0.555
Static Until Stable	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
Static Model	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
MRE for Smoothed Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
$K = 0.25$	0.035	0.004	0.194	0.338	5.656	0.269	0.214	1.833
$K = 0.50$	0.785	0.841	1.154	1.478	11.780	0.325	1.192	4.351
$K = 0.75$	1.459	1.508	1.910	2.407	16.919	0.774	1.945	6.441
Linear K	0.828	0.784	0.649	0.855	7.218	0.312	1.292	1.980
$K = 1 - L^2$	0.829	0.901	0.869	1.300	9.107	0.515	1.694	2.533
$K = e^{-L}$	0.866	0.915	0.943	1.328	10.046	0.384	1.610	3.076
3 Step Function	0.857	0.868	0.889	1.247	6.714	0.453	1.135	1.597
5 Step Function	0.813	0.722	0.719	0.782	0.474	0.759	0.515	0.873
Static Until Stable	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Static Model	0.401	0.452	0.581	0.735	0.313	0.759	0.515	0.880
SRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
$K = 0.25$	1.886	1.640	1.631	2.166	28.493	1.156	2.188	5.840
$K = 0.50$	3.518	3.071	3.037	4.060	54.226	2.115	4.137	11.136
$K = 0.75$	4.580	3.978	3.882	5.305	74.251	2.602	5.451	15.182
Linear K	1.343	2.124	1.578	2.711	27.746	1.358	3.313	9.282
$K = 1 - L^2$	1.343	1.927	1.625	3.033	29.022	1.416	3.777	9.743
$K = e^{-L}$	1.712	2.530	2.143	3.376	33.252	1.693	4.132	10.577
3 Step Function	0.779	1.275	1.665	2.483	17.180	1.440	3.139	5.544
5 Step Function	0.116	0.384	0.475	0.564	0.030	0.554	0.545	0.161
Static Until Stable	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
Static Model	0.594	0.658	0.832	0.913	0.269	0.807	0.638	0.460
SRE for Smoothed Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
$K = 0.25$	1.138	0.436	0.500	0.860	21.212	0.333	0.540	3.236
$K = 0.50$	2.922	1.629	1.700	2.436	41.375	0.987	1.788	6.981
$K = 0.75$	4.271	2.537	2.594	3.664	57.516	1.639	2.722	9.994
Linear K	1.340	1.033	0.941	1.168	36.615	0.570	1.942	4.010
$K = 1 - L^2$	1.877	1.524	1.409	2.062	36.397	0.961	2.364	5.473
$K = e^{-L}$	2.302	1.612	1.427	2.301	43.005	1.033	2.326	6.332
3 Step Function	2.228	1.478	1.344	2.049	21.497	1.038	1.546	3.053
5 Step Function	0.863	0.775	0.853	0.874	0.404	0.772	0.584	0.622
Static Until Stable	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
Static Model	0.549	0.564	0.713	0.844	0.266	0.772	0.584	0.624

Table 4.7: Weighted Average Method For Logarithmic Model, Indirect Estimation

MRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
$K = 0.25$	0.251	0.225	0.253	0.270	0.358	0.143	0.209	0.349
$K = 0.50$	0.844	0.741	0.767	0.820	1.117	0.403	0.654	1.037
$K = 0.75$	1.301	1.138	1.153	1.239	1.735	0.645	1.011	1.586
Linear K	0.431	0.370	0.377	0.537	0.524	0.417	0.538	0.665
$K = 1 - L^2$	0.448	0.434	0.416	0.664	0.557	0.458	0.680	0.796
$K = e^{-L}$	0.452	0.489	0.485	0.660	0.558	0.475	0.671	0.787
3 Step Function	0.363	0.344	0.370	0.488	0.446	0.478	0.537	0.530
5 Step Function	0.354	0.257	0.207	0.146	0.362	0.518	0.130	0.338
Static Until Stable	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
Static Model	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
MRE for Smoothed Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
$K = 0.25$	0.034	0.134	0.123	0.051	0.026	0.268	0.184	0.020
$K = 0.50$	0.787	0.577	0.550	0.731	0.925	0.326	0.436	0.862
$K = 0.75$	1.462	1.144	1.078	1.367	1.678	0.776	0.911	1.559
Linear K	0.828	0.748	0.601	0.586	0.825	0.313	0.603	0.768
$K = 1 - L^2$	0.829	0.791	0.691	0.838	1.028	0.517	0.785	0.952
$K = e^{-L}$	0.866	0.804	0.659	0.813	1.087	0.385	0.700	0.900
3 Step Function	0.857	0.763	0.622	0.756	0.745	0.454	0.438	0.561
5 Step Function	0.807	0.678	0.548	0.381	0.336	0.579	0.122	0.636
Static Until Stable	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Static Model	0.371	0.342	0.239	0.132	0.126	0.579	0.122	0.484
SRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
$K = 0.25$	1.889	1.355	1.068	1.405	4.006	1.157	1.296	1.788
$K = 0.50$	3.523	2.539	1.995	2.639	7.614	2.118	2.465	3.434
$K = 0.75$	4.586	3.279	2.528	3.429	10.384	2.608	3.234	4.664
Linear K	1.345	1.780	1.058	1.774	4.058	1.361	1.977	2.969
$K = 1 - L^2$	1.345	1.604	1.086	1.970	4.240	1.423	2.239	3.116
$K = e^{-L}$	1.714	2.116	1.435	2.216	4.864	1.697	2.468	3.373
3 Step Function	0.778	1.065	1.132	1.616	2.647	1.414	1.857	2.065
5 Step Function	0.114	0.324	0.374	0.296	0.314	0.438	0.180	0.766
Static Until Stable	0.021	0.044	0.050	0.045	0.006	0.036	0.067	0.061
Static Model	0.563	0.527	0.567	0.435	0.915	0.616	0.212	1.014
SRE for Smoothed Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
$K = 0.25$	1.140	0.303	0.181	0.347	2.369	0.333	0.192	0.513
$K = 0.50$	2.926	1.252	0.940	1.397	5.348	0.989	0.822	1.769
$K = 0.75$	4.276	2.023	1.566	2.240	7.729	1.642	1.414	2.778
Linear K	1.341	0.880	0.713	0.636	5.170	0.571	1.044	1.354
$K = 1 - L^2$	1.879	1.297	1.042	1.250	5.114	0.964	1.226	1.817
$K = e^{-L}$	2.304	1.372	1.055	1.416	5.999	1.035	1.163	1.844
3 Step Function	2.230	1.258	0.996	1.242	3.211	1.040	0.691	1.037
5 Step Function	0.858	0.699	0.737	0.441	0.910	0.569	0.111	0.830
Static Until Stable	0.875	0.895	0.848	0.885	0.898	0.917	0.862	0.864
Static Model	0.518	0.427	0.381	0.290	0.846	0.569	0.111	0.733

4.4.2 Analysis of Initial Experiments

One of the first things that is apparent examining the data in tables 4.2 through 4.4 is that stabilization is of no help in improving short term predictions, as manifest by SRE scores which are higher when stabilization is applied than when it is not. This is not surprising, testing is rarely random and errors are likely to cluster. Any technique which works to remove noise from model predictions is also likely to stretch errors out to occur more uniformly, thus reducing short term accuracy. This observation holds for both the exponential and the logarithmic models, and for all variations on both the replacement and weighted average methods.

Examining the results for the use of the replacement method on the exponential model, we find some improvement. In the case of the replacement method, the best results seem to be evenly split between the replacement of only β_0 and the dual replacement variation. The technique of allowing each parameter to be replaced independently produces similar results when the data is smoothed, presumably smoothing creates a situation where if one parameters exceeds the legal values then the other will as well. The weighted average method also seems to produce good results for the exponential model, providing smoothing is not applied. For many data sets, many of the weighting schemes examined appear to offer some improvement, with $K = 0.75$ and the five step function appearing to be particularly promising. However, this only holds when smoothing is not applied. Although the five step function combined with smoothing offers some improvement for some data sets, it appears that smoothing generally hinders the weighted average method. Even when it gives results that are better than when no enhancements are applied the same weighting scheme without smoothing does better. This observation is consistent with our previous observation in chapter 2, that smoothing makes it more difficult for the maximum likelihood method to produce accurate long term predictions.

Stabilization also appears promising for the logarithmic model. The independent and dual replacement methods give identical results, suggesting that whenever one parameter exceeds the acceptable range, the other almost always did as well; just as in the exponential case. The weighted average method also gave good results, particularly when K was set to a constant value of 0.25. Unlike the case for the exponential model, smoothing does appear to have some benefit here, with the best long term accuracy scores being obtained when the data was smoothed. Comparing direct estimation to the indirect estimation we find that in the majority of cases the indirect method provided better results. The direct method shows some promise, but it is apparent that at this time we can best estimate the parameters of the logarithmic model by estimating the parameters for the exponential model and transforming them.

Several things can be deduced by examining our results. First, stabilization can potentially improve parameter estimation when good static parameter estimates can be obtained. In the case of the exponential model we knew, or where able to closely approximate, all values required to compute the static estimates. Using these nearly perfect static estimates we obtained very good results using stabilization. Using these parameter estimates for the exponential model we were also able to derive estimates for the logarithmic model, and our results in this case are also quite encouraging. This suggests that our method of relating the parameters of the two cases is accurate, providing experimental evidence to support our previous assertions. Results from the direct method of parameter estimation suggest that with some refinement this method may also be effective. Variation due to uncertainties about source size and processor speed may have effected our estimates, and it may be that we need better methods for setting the value of D_{min} and K_{min} .

What we can not conclude based on these experiments is that stabilization is effective in practice. The two cases where it was effective, the exponential model and the logarithmic model using static parameters based on the estimates for the exponential model, based the static parameters on known values for the defect density and fault exposure ratio. As previously explained, accurate estimation of the defect density should be possible provided enough data is available from previous projects by the same organization. However, because we are using known values for K which would not be available in practice we can not conclusively show that stabilization has practical value. In order to demonstrate this we must conduct a second round of tests, using an estimated value for K .

4.4.3 Second Round of Experiments

It has been suggest by Malaiya and Denton (Malaiya and Denton 1997) that the fault exposure ratio might be estimated as given in equation 4.6. Using this equation to estimate K requires that the defect density be known, but as discussed previously, there are a number of techniques which provide estimates of defect density.

$$K = \frac{1.2 \times 10^{-6}}{D_0} e^{0.05D_0} \quad (4.6)$$

Using this estimated value for the fault exposure ratio, we can again obtain estimates for the static parameters of the exponential and logarithmic models. We use the same calculations as before, substituting only the value for K . Table 4.8 gives the values for the statically estimated parameters, and also the estimated value of K used to obtain these values. Note that β_0 remains the same, as it is dependent on only the estimated defect density; it is only β_1 that changes due to its dependence on the fault exposure ratio.

Table 4.8: Static Parameters Used by Data Set, Estimated K

Data Set	K	Exponential Model		Logarithmic Model	
		β_0	β_1	β_0	β_1
T1	0.000000166	142.800	0.00000765	47.649	0.00005027
T2	0.000000303	56.700	0.00001093	18.920	0.00007183
T3	0.000000348	39.900	0.00001489	13.314	0.00009780
T4	0.000000356	55.650	0.00001062	18.569	0.00006974
T5	0.000001406	872.550	0.00000058	291.151	0.00000378
T6	0.000000192	76.650	0.00003363	25.576	0.00022093
T17	0.000000376	39.900	0.00000608	13.314	0.00003995
T27	0.000000649	43.050	0.00000515	14.365	0.00003381

Using these new values for the static parameters we repeat our earlier experiments. Because stabilization was not found to be effective in reducing the error in short term predictions, we now examine only long term model accuracy. Further, we examine only the exponential model and the logarithmic model using parameter estimates based on the estimates for the exponential model. The direct method of model estimation does not require that the fault exposure ratio be known, and is thus not subject to the criticism which necessitate this second round of experiments. In any event, our results in the first round suggest that more work in other areas is required before direct estimation becomes the preferred method of setting the static parameters for the logarithmic model.

Table 4.9 presents the results of the replacement methods on the exponential model, table 4.10 presents the results for the different weighted average methods. Those entries in bold represent the best results obtained for a given data set, taking into consideration all stabilization techniques, the absolute best results obtained. Entries in italics denote the sub-technique which produced the best results for the method in question. For instance, the technique where each parameter can be replaced independently produced good results for the logarithmic model on the first three data sets. These results were better than any of the other replacement techniques, but not as good as the best results obtained using the weighted averages technique. Consequently, we show the entries for this method and these data sets in italics.

Several things are apparent examining table 4.9. First, for six of the eight data sets examined the replacement method produces the best possible results. Table 4.10 shows that for the T1 data set the difference between the replacement method and weighted average method is very small and for the T6 data set the difference, although not as small, is also not large, suggesting that overall the replacement method is the clearly superior technique. For half the data sets in question the best variation on the replacement method is to make only β_0 subject to replacement, and to employ no smoothing on the data in question. Although the dual replacement method provides significantly better results in two cases the results of this technique are much worse in other cases, where as the results for the first technique offer a consistent improvement for all data sets. As table 4.10 shows, the weighted average method also has potential, particularly when a high constant weight is assigned to the static parameters. However, overall, the replacement technique appears to be the better method.

It is worth noting that in only one case, for the replacement method on the T1 data set, did smoothing prove useful. This is consistent with our previous observation that smoothing does not aid in the finding of accurate parameter estimates for the exponential model. Also worth noting is that a comparison between the results of the first round of experiments and the second shows that the numbers do not vary greatly. Although a direct comparison shows that stabilization produced results that were marginally better when using exact values of

K in most cases, in a few instances the estimated value of K gave better results. This supports our findings that replacement of β_0 is the best method of stabilization; the parameter β_1 which is dependent upon K is not as significant as β_0 which is not dependent on K . It also is in line with our previous observation that the maximum likelihood method produces estimates of β_0 which mirror the number of faults already found. By replacing early estimates which are far to low, and allowing only β_1 to be set dynamically in the early stages we are able to produce more reliable estimates of long term reliability.

Table 4.9: Results of Parameter Replacement with the Exponential Model, Estimated K

	T1	T2	T3	T4	T5	T6	T17	T27
MRE for Raw Data								
No Stabilization	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
Replacement of β_0	0.274	0.151	0.163	0.168	0.140	0.156	0.172	0.168
Replacement of β_1	0.800	0.956	2.047	1.482	0.507	0.999	1.316	1.776
Independent	0.550	0.366	0.428	0.613	0.139	0.856	0.315	0.141
Dual Replacement	0.483	0.270	0.335	0.552	0.050	0.835	0.203	0.049
MRE for Smooth Data								
No Stabilization	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
Replacement of β_0	0.095	0.237	0.202	0.205	0.061	0.207	0.207	0.237
Replacement of β_1	1.248	7.547	6.700	4.899	0.971	2.624	14.215	14.275
Independent	0.483	0.270	0.335	0.552	0.050	0.835	0.203	0.050
Dual Replacement	0.483	0.270	0.335	0.552	0.050	0.835	0.203	0.050

The results of using stabilization on the logarithmic method are somewhat different. Tables 4.11 and 4.12 present the results of our second round of experiments with the logarithmic model. Comparing these tables to 4.4 and 4.7 there seems to be little difference between the results of the first round and the second. Where a difference does exist it seems to be evenly split between favoring an exact value of K and an estimated value. This suggests that knowing the exact fault exposure ratio is even less important to accurate use of the logarithmic model than it is to the exponential model, provided some reasonable value is available. Other factors, such as the minimum defect density, probably play a much greater role in this model. It is also worth noting that in table 4.11 the results for independent and dual replacement are identical to three significant digits, for both raw and smooth data. This suggests not only that model parameters tend to exceed that predicted range together, but that smoothing has little to no effect when this method of stabilization is used.

Smoothing does have a significant effect when the weighted average method of stabilization is used. In contrast to our results with the exponential model we find that the weighted average method on smoothed data provides the best improvement in long term predictive accuracy. Examining table 4.12 we find that a low constant value of K ($K = 0.25$), combined with data smoothing, results in a consistent and dramatic increase in model accuracy. Again, for six of the eight data sets this method results in the best improvement. In the case of the T6 data set the best results are obtained on raw data with $K = 0.25$, and in case of the T17 data set the difference between the smoothing with $K = 0.25$ and the best method is very small.

A comparison between the best results from stabilization and the results of running the models without stabilization shows that in some cases an order of magnitude increase in model accuracy was obtained. More commonly stabilization resulted in MRE scores that were less than half of those obtained without stabilization. This represents a substantial improvement, indicating that overall stabilization reduced the error in estimating the number of faults present at the end of testing by more than half. The best comparative results for the logarithmic model come from using the weighted average technique with data smoothing. It should be noted that smoothing reduces the number of data points in a data set. This results in an error measure which is slightly different, although this effect is minimized because we average the error over the number of data points in the set for which we are able to make valid parameter estimates. We feel that the resulting improvement in this case is more than can be accounted for by this reduction in the size of the data set only.

Table 4.13 presents a summary of the results in the second round of experiments. It lists the various techniques which produced the the best overall long term accuracy scores, and gives the number of times in which they succeeded in obtaining the best results, broken down for the logarithmic and exponential models.

Table 4.10: Weighted Average Method For The Exponential Model, Estimated K

MRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
$K = 0.25$	0.809	0.484	0.995	0.422	0.368	0.379	0.550	1.316
$K = 0.50$	0.305	0.333	0.743	0.327	0.230	0.221	0.458	0.865
$K = 0.75$	0.093	<i>0.155</i>	0.390	<i>0.220</i>	<i>0.095</i>	0.109	0.249	0.413
Linear K	0.813	0.323	0.986	0.370	0.423	0.309	0.384	0.547
$K = 1 - L^2$	0.508	0.249	0.928	0.252	0.400	0.273	0.195	0.247
$K = e^{-L}$	0.506	0.270	0.848	0.317	0.409	0.278	0.314	0.473
3 Step Function	0.721	0.280	0.921	0.286	0.406	0.409	<i>0.192</i>	0.202
5 Step Function	0.440	0.259	<i>0.303</i>	0.429	0.377	0.707	0.205	<i>0.121</i>
Static Until Stable	2.163	0.599	1.117	1.011	0.498	0.586	0.449	0.601
Static Model	0.483	0.270	0.335	0.552	0.050	0.835	0.203	0.050

MRE for Smooth Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
$K = 0.25$	0.785	2.346	2.122	1.447	0.714	0.817	4.430	10.659
$K = 0.50$	0.520	2.469	2.164	1.461	0.461	0.736	4.801	7.122
$K = 0.75$	0.233	1.560	1.368	0.931	0.206	0.472	3.049	3.546
Linear K	0.990	2.037	2.179	1.081	0.642	0.536	4.817	10.736
$K = 1 - L^2$	0.961	2.586	2.457	1.289	0.520	0.524	3.247	8.202
$K = e^{-L}$	0.928	2.765	2.427	1.583	0.522	0.687	4.290	7.674
3 Step Function	0.936	2.648	2.343	1.511	0.525	0.622	3.799	9.559
5 Step Function	0.835	0.652	0.601	0.660	0.280	0.835	0.203	0.376
Static Until Stable	1.021	0.803	0.880	0.759	0.945	0.909	0.836	0.756
Static Model	0.483	0.270	0.335	0.552	0.050	0.835	0.203	0.050

Table 4.11: Results of Parameter Replacement with the Logarithmic Model, Estimated K

	T1	T2	T3	T4	T5	T6	T17	T27
MRE for Raw Data								
No Stabilization	0.412	0.393	0.343	<i>0.364</i>	<i>0.468</i>	<i>0.414</i>	0.399	<i>0.405</i>
Replacement of β_0	2.386	2.164	2.170	2.288	2.977	1.519	1.997	2.759
Replacement of β_1	0.895	0.852	0.851	0.893	0.791	0.937	0.834	0.720
Independent	<i>0.405</i>	<i>0.238</i>	<i>0.290</i>	0.461	0.540	0.736	0.182	0.748
Dual Replacement	0.405	0.238	0.290	0.461	0.540	0.736	0.182	0.748
MRE for Smooth Data								
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Replacement of β_0	2.746	2.320	2.217	2.614	3.040	1.834	1.996	2.865
Replacement of β_1	0.988	0.981	0.970	0.985	0.978	0.994	0.965	0.951
Independent	0.405	0.238	0.290	0.461	0.540	0.736	0.182	0.748
Dual Replacement	0.405	0.238	0.290	0.461	0.540	0.736	0.182	0.748

Table 4.12: Weighted Average Method For Logarithmic Model, Estimated K

MRE for Raw Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
$K = 0.25$	0.251	0.225	0.253	0.270	0.358	0.143	0.209	0.349
$K = 0.50$	0.844	0.741	0.767	0.820	1.117	0.402	0.654	1.037
$K = 0.75$	1.301	1.138	1.153	1.238	1.736	0.644	1.011	1.587
Linear K	0.431	0.370	0.377	0.537	0.524	0.417	0.538	0.670
$K = 1 - L^2$	0.448	0.435	0.416	0.663	0.559	0.456	0.681	0.805
$K = e^{-L}$	0.452	0.489	0.485	0.660	0.558	0.474	0.670	0.791
3 Step Function	0.365	0.330	0.374	0.532	0.471	0.504	0.546	0.583
5 Step Function	0.358	0.208	0.232	0.336	0.423	0.621	0.179	0.483
Static Until Stable	0.412	0.393	0.343	0.364	0.468	0.414	0.399	0.405
Static Model	0.405	0.238	0.290	0.461	0.540	0.736	0.182	0.748
MRE for Smoothed Data								
K function	T1	T2	T3	T4	T5	T6	T17	T27
No Stabilization	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
$K = 0.25$	0.034	0.134	0.123	0.051	0.026	0.268	0.184	0.020
$K = 0.50$	0.787	0.578	0.550	0.731	0.925	0.326	0.435	0.862
$K = 0.75$	1.462	1.144	1.078	1.367	1.678	0.776	0.911	1.560
Linear K	0.828	0.748	0.601	0.586	0.825	0.313	0.603	0.768
$K = 1 - L^2$	0.829	0.791	0.691	0.838	1.038	0.516	0.784	0.953
$K = e^{-L}$	0.866	0.804	0.659	0.812	1.087	0.385	0.700	0.900
3 Step Function	0.857	0.763	0.622	0.756	0.813	0.453	0.458	0.614
5 Step Function	0.814	0.636	0.574	0.600	0.642	0.736	0.182	0.794
Static Until Stable	0.925	0.916	0.865	0.903	0.944	0.933	0.873	0.892
Static Model	0.405	0.238	0.290	0.461	0.540	0.736	0.182	0.748

Table 4.13: Summary of Stabilization Results

Technique	Exp. Model	Log. Model
Replacement of β_0 , Raw Data	4	0
Dual Replacement, Raw Data	2	1
$K = 0.75$, Raw Data	2	0
$K = 0.25$, Raw Data	0	1
$K = 0.25$, Smooth Data	0	6

4.5 Conclusion

Stabilization is an effective technique, if properly applied. What constitutes proper application differs between the exponential and logarithmic models. The exponential model does best when stabilization is applied to raw data; and when static estimates of model parameters are used to set acceptable bounds on β_0 and replace this parameter when the dynamically fitted values fall outside the acceptable range. The logarithmic model performs best when its input data is subjected to two rounds of lump smoothing, and the parameters used are a weighted average of parameters with the statically fitted values providing 25% of the final value and the dynamically fitted values providing the rest. In this case, the statically fitted values should be found by first estimating the parameters for the exponential model and then using these values to find the values for the logarithmic model parameters based on the relationship between the two models. Stabilization provides no benefits for short term predictions.

Stabilization pre-supposes that reasonably accurate estimates for the static parameters can be obtained. Obtaining such estimates requires that the initial defect density, linear execution time, and fault exposure ratio of the software in question all be known. Linear execution time should be available as soon as running code for the program exists, a simple count of object instructions divided by the MIPS rate of the testing processor should yield this value. An accurate estimate of defect density is critical to setting the value of β_0 , which appears to be the most important parameter. Numerous methods exist to aid in the estimation of defect density prior to the start of testing or relatively early on in the testing process; and we present a new one in chapter 5. The best method may be to base estimates on the experience the developing organization has with the type of software in question. The fault exposure ratio may be more difficult to estimate accurately. However, it is possible to estimate it based on the defect density, and this method appears to be reasonably effective. It is also important to note the since β_0 appears to be the most important parameter in obtaining long term projections and it is based solely of defect density, accurate estimate of the fault exposure ratio may not be highly significant to the process.

Chapter 5

A COVERAGE BASED MODEL

5.1 Introduction

The estimated number of remaining defects in a piece of code is often used as an acceptance criterion. Sources in the literature suggest that leading edge software development organizations typically achieve a defect density of about 2.0 defects per thousand lines of code. (Binder 1997); it has been suggested that some organizations now target even lower defect densities. The NASA Space Shuttle Avionics software, with an estimated defect density of 0.1 defects per thousand lines (KLOC), is regarded as an example of what can be currently achieved by the best methods (Hatton 1997), at a cost of \$1000 per line of code. Unfortunately fixing defects in the field can cost several orders of magnitude more than fixing a defect prior to release, but a program must be released by some deadline. Consequently, accurate measurements of expected software defect density are important in determining if a piece of software is reliable enough to release.

One conceivable way of knowing the exact defect density of a program is to actually find all remaining defects. This is obviously infeasible for any commercial product. Even if resources are available, it will take a prohibitive amount of time to find all bugs in a large program (Butler and Finelli 1993). Sampling based methods have been suggested for estimating the number of remaining defects. McConnell (McConnell 1997) has given a method that involves using two independent testing activities, perhaps by two different teams of testers. This and other sampling techniques assume that faults found have the same testability as faults not found. However, in actual practice, the faults not found represent faults that are harder to find (Malaiya and Yang 1984). Thus, such techniques are likely to yield an estimate of faults that are relatively easier to find and thus less than the true number. Fault seeding methods (McConnell 1997) suffer from similar problems.

It is possible to estimate the defect density based on past experience using empirical models like the Rome Lab model (Lahey and Neufelder 1997) or the model proposed by Malaiya and Denton (Malaiya and Denton 1997). The estimates obtained by such models can be very useful for initial planning, however these models are not expected to be accurate enough to compare with methods involving actual test data.

An alternative is to estimate the number of faults by using the exponential SRGM. As discussed in chapter 3, the parameter β_0 of the exponential model can be taken as an estimate of the total number of faults present in the system. This suggests that we can estimate the remaining faults by fitting the exponential model to the data available and subtracting the number of faults found from β_0^E . We take the number of test cases applied as the measure of time, and apply this technique to data from Pasquini. Figure 5.1 shows the results, which are typical for this technique. Several things are worth noting about this plot. First, testing continued to reveal defects even after this technique predicted that there were no remaining defects. Second, the value of β_0^E never stabilizes. Towards the end of testing β_0^E takes on a value very close to the number of defects already found, and thus provides no useful information.

In hindsight, the difficulty encountered by the exponential model is not unexpected. A traditional SRGM relates the number of defects found to the testing time spent; but the number of defects found can vary based on test effectiveness, processor speed, and other factors. This difficulty could be resolved if we had a model that based the expected number of defects on an objective measure of test effectiveness, such as coverage of program enumerables like blocks or branches. The relationship between test coverage and the number of defects found has been investigated by Piwowarski, Ohba and Caruso (Piwowarski, Ohba, and Caruso 1993),

Hutchins, Goradia and Ostrand (Hutchings, Goradia, and Ostrand 1994), Malaiya et al (Malaiya, Li, Bieman, Karcich, and Skibbe 1994), Lyu, Horgan and London (Lyu, Horgan, and London 1993), and Chen, Lyu and Wong (Chen, Lyu, and Wong 1996).

In the next two sections, a model for defect density in terms of test coverage is introduced and its applicability is demonstrated using test data. Section 5.4 presents some thought on the meaning of the parameters of this model, and section 5.4.1 presents a two-parameter approximation of the model and extends the meaning of the parameters to this case. Finally we present some observations on this new approach. Portions the work presented here appeared as “Estimating The Number of Residual Defects” by Malaiya and Denton in (Malaiya and Denton 1998c).

5.2 A Coverage based approach

Recently, a model was presented by Malaiya et al that relates the number of residual defects with test coverage measures (Malaiya, Li, Bieman, Karcich, and Skibbe 1994). This model is based on the logarithmic Poisson SRGM which has been found to have superior predictive capabilities (Malaiya, Karunanithi, and Verma 1992). We assume that the Logarithmic model is applicable to the total number of defects found. Using the Logarithmic Poisson model we can replace time with test coverage to obtain a new model. We assume that coverage units like branches etc. also follow a Logarithmic Poisson model with respect to testing time. This assumption is based on the fact that different branches etc. have different probabilities of getting covered just like defects. Here we use the superscript D for defects and $i = 1, 2, \dots$ for various test units like blocks, branches etc. Using the Logarithmic Poisson model we can then express defect coverage $C^D(t)$ as,

$$C^D(t) = \frac{\beta_0^D}{N_0^D} \ln(1 + \beta_1^D t) \quad C^D(t) \leq 1 \quad (5.1)$$

Similarly for test unit i coverage, let us assume,

$$C^i(t) = \frac{\beta_0^i}{N_0^i} \ln(1 + \beta_1^i t) \quad C^i(t) \leq 1 \quad (5.2)$$

where N_0^D is the total initial number of defects and N^i is the total number of units of type i in the program under test. Here $\beta_0^D, \beta_1^D, \beta_0^i, \beta_1^i$, are appropriate parameters for the applicable Logarithmic Poisson model; with respect to either defects or the appropriate enumerable.

We can solve for t using equation 5.2 and substitute it in to equation 5.1 to obtain

$$C^D(C^i) = a_0^i \ln[1 + a_1^i (e^{a_2^i C^i} - 1)] \quad C^i \leq 1 \quad (5.3)$$

where

$$a_0^i = \frac{\beta_0^D}{N_0^D} \quad (5.4)$$

$$a_1^i = \frac{\beta_1^D}{\beta_1^i} \quad (5.5)$$

$$a_2^i = \frac{N^i}{\beta_0^i} \quad (5.6)$$

The previous equations give us defect coverage from enumerable coverage. Although enumerable coverage can easily be obtained from a variety of testing tools, defect coverage is harder to interpret. Rather than have the model return a percentage of defects covered, we would prefer it to give us the actual number of defects that would be expected at a given coverage level. If we indicate the total expected number of defects found in time t by $\mu(t)$, we can write $C^D(t) = \frac{\mu(t)^D}{N_0^D}$. Hence from equation 5.3,

$$\mu^D(C^i) = a_3^i \ln[1 + a_1^i (e^{a_2^i C^i} - 1)], \quad C^i \leq 1 \quad (5.7)$$

where $a_3^i = a_0^i \times N_0^D = \beta_0^D$

We must note that equations 5.3 and 5.7 are applicable only when C^i is less than or equal to one. Note that equation 5.7 allows for less than 100% defect coverage when enumerable coverage is at 100%. This is because exercising all branches does not exercise the program exhaustively. Achieving 100% coverage using a more strict measure, such as P-uses, would exercise the code more thoroughly but again not necessarily cover all defects.

Figure 5.2 shows a plot illustrating the shape of the curve described by equation 5.3. At the beginning, defect coverage grows only slowly with test unit coverage. However, at higher test coverage, there is a linear relationship. The value around which the curve exhibits a knee has a significance as we will see below. The important point is to note that at 100% coverage of unit i , we expect to find the number of defects given by the point where the curve intersects the vertical line corresponding to 100% coverage. We can expect that at least this many faults were initially present. A lower bound on the number of remaining faults is obtained by subtracting the number of faults actually found from the number of faults expected at 100% coverage. The bound is closer to the actual number if we use a more strict test coverage measure.

The numbers across the top of the plot in figure 5.2 give the number of test cases applied for the top curve. They show that despite the linear growth of defect coverage, more testing effort is needed to obtain more coverage and thus find more faults. The second curve shows the plot that would be obtained if some of the faults were already removed in a previous test phase when current testing was initiated. Figure 5.2 shows that when the initial defect density is lower, testing starts finding defects at a higher coverage level. This corresponds to shifting of the curve downwards.

5.3 Applying the New Approach

Applicability of this model is illustrated by the plots in figures 5.3, 5.4, and 5.5. All these plots show enumerable coverage against expected defects found, showing both the actual test data and the fitted model. This data was collected experimentally by Pasquini et al from a 6100 line C program by applying 20,000 test cases (Pasquini, Crespo, and Matrella 1996). The test coverage data was collected using the ATAC tool. Figure 5.3 shows a screen from ROBUST, a tool we have developed which implements our technique; see chapter 6 for details of this program.

For the 20,000 tests, the coverage values obtained were: block coverage : 82.31% of 2970 blocks, decision coverage : 70.71% of 1171 decisions, and p-use coverage 61.51% of 2546 p-uses. This is to be expected since p-use coverage is the most rigorous coverage measure and block coverage is the least. Complete branch coverage guarantees complete block coverage, and complete p-use coverage guarantees complete branch coverage (Frankl and Weyuker 1988).

The plots suggest that 100% block coverage would uncover 40 defects, 100% branch coverage would uncover 47 defects, and 100% p-use coverage would reveal 51 defects. If we assume that all of the software components are reachable, then we can expect the total number of faults to be slightly more than 51. In actual practice, large programs often contain some unreachable code, in the form of dead or obsolete code. For C programs this can be in the neighborhood of 5% of the total code (Srivastava 1992). For accurate estimates this needs to be taken into account. For simplicity of illustration, here we will assume that all of the code is reachable. Unreachable code can be minimized by using coverage tools and making sure that all modules and sections are entered during testing. The data sets presented here are for programs that are not evolving and thus the actual defect density is constant.

The coverage model in Equations 5.3 and 5.7 provides us a new way to estimate the total number of defects. As we can see in Figures 5.3, 5.4 and 5.5, which use the data obtained by Pasquini et al., the data points follow the linear part of the model rather closely. Table 5.1 shows gives the number of defects we would expect to find for the Pasquini data, if 100% coverage of each enumerable was obtained. It also gives the coverage actually obtained, and the number of faults that were found at this level.

For this project 1240 tests revealed 28 faults. Another 18,760 tests did not find any additional faults, even though at least five more faults were known to exist. The data suggests that the enumerables (blocks, branches

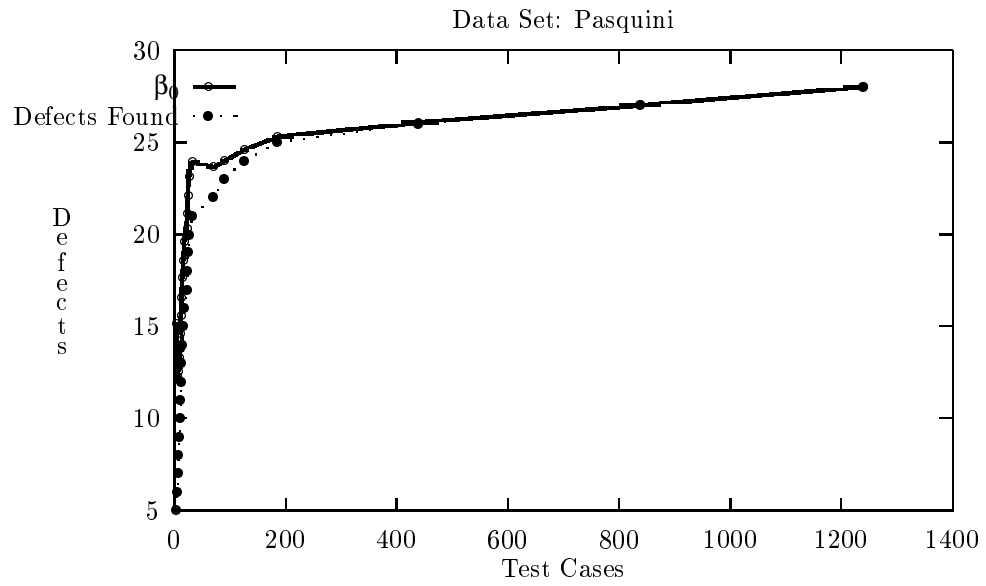


Figure 5.1: Estimated total defects using exponential model (Pasquini data)

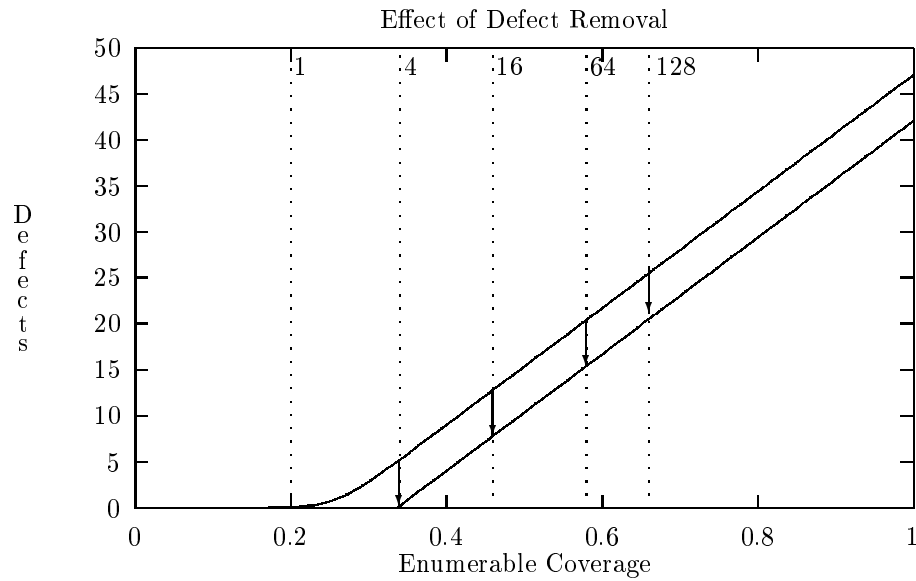


Figure 5.2: Defects vs. Test Coverage Model

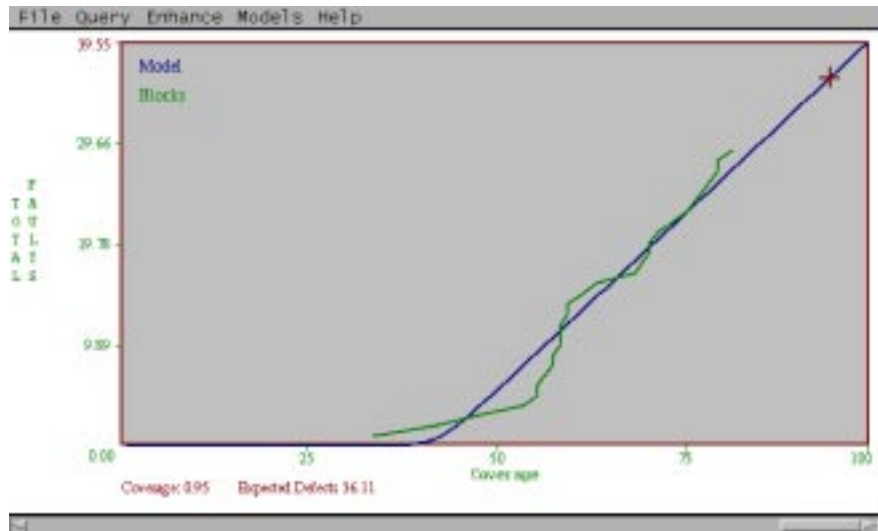


Figure 5.3: ROBUST: Block Coverage data (Pasquini et al.)

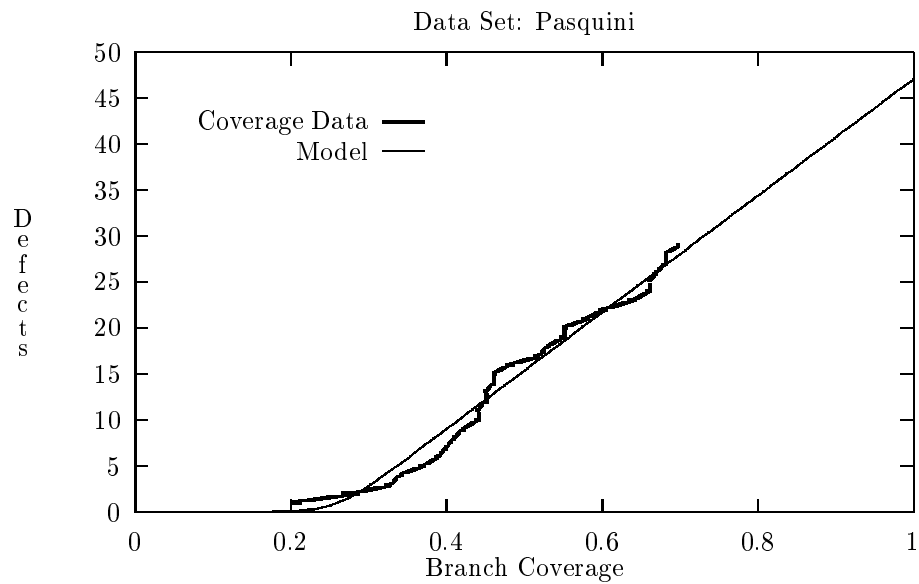


Figure 5.4: Defects vs. % Branch Coverage

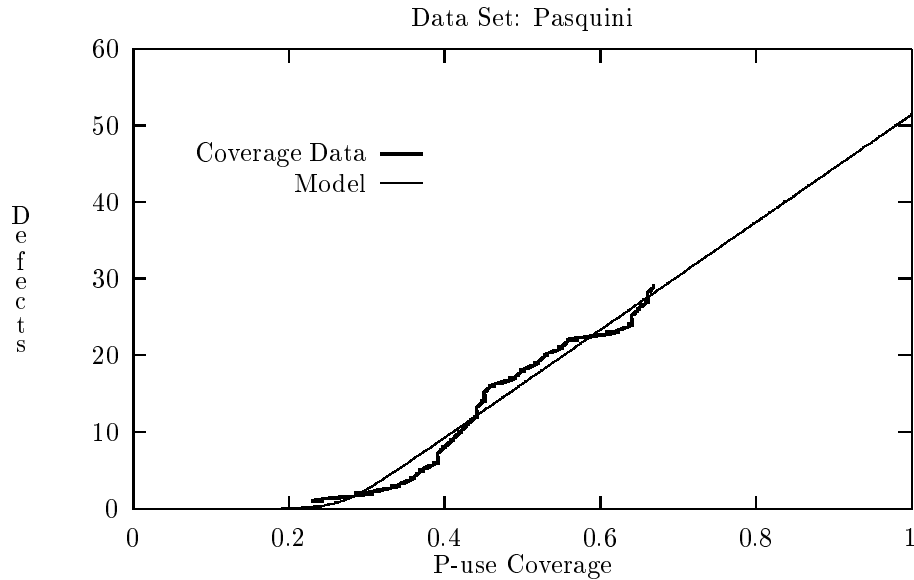


Figure 5.5: Defects vs. % P-use Coverage

Table 5.1: Projected number of total defects with 100% coverage

Coverage Measure	Defects Found	Defects Expected	Coverage Achieved	Defects at 100%
Block Coverage	28	27.2	82%	40
Branch Coverage	28	28.0	70%	47
P-uses Coverage	28	28.6	67%	51
C-uses Coverage	28	26.2	74%	43

etc.) not covered by the first 1240 tests were very hard to reach. They perhaps belong to sections of the code intended for handling special situations or error conditions. It is also possible that they represent dead code of some form.

Because there is a subsumption relationship between blocks, branches, and p-uses we know that all defects found by 100% block coverage will also be found by 100% branch coverage, and all defects found by 100% branch coverage will also be found by 100% p-use coverage. We thus say that p-uses is the strictest coverage measure, because it will cover the largest portion of the code under test. There is no coverage measure such that 100% coverage will assure detection of all the defects. Thus in the above table, the entry in the last column is a low estimate of the total number of defects actually present. Using a more strict coverage measure raises the low estimate closer to the actual value. Thus the estimate of 51 faults using P-use coverage should be closer to the actual number than the estimates provided by block coverage. Two coverage measures, DU-path coverage and all-path coverage are more strict than P-use coverage (Frankl and Weyuker 1988), and may be suitable for cases where ultra-high reliability is required. However considering the fact that often even obtaining 100% branch coverage is infeasible for many programs, we are unlikely to detect more faults than what the P-use coverage data provides even with rigorous testing. It should be noted that C-use coverage does not fit in the subsumption hierarchy and therefore it is hard to interpret the values obtained by using C-use coverage data.

Further application of this new method is illustrated by examining the data provided by Vouk (Vouk 1992). These three data sets were obtained by testing three separate implementations of a sensor management program for an inertial navigation system. Each program is about five thousand lines of code. In the first program, 1196 tests found 9 defects. For the other two programs 796 test revealed 7 and 9 defects respectively. Figure 5.6 shows the plots of P-use coverage achieved versus the number of defects found. Table 5.2 shows the estimates for the total number of faults that would be found with 100% coverage, which is again consistent with the subsumption hierarchy between coverage enumerables.

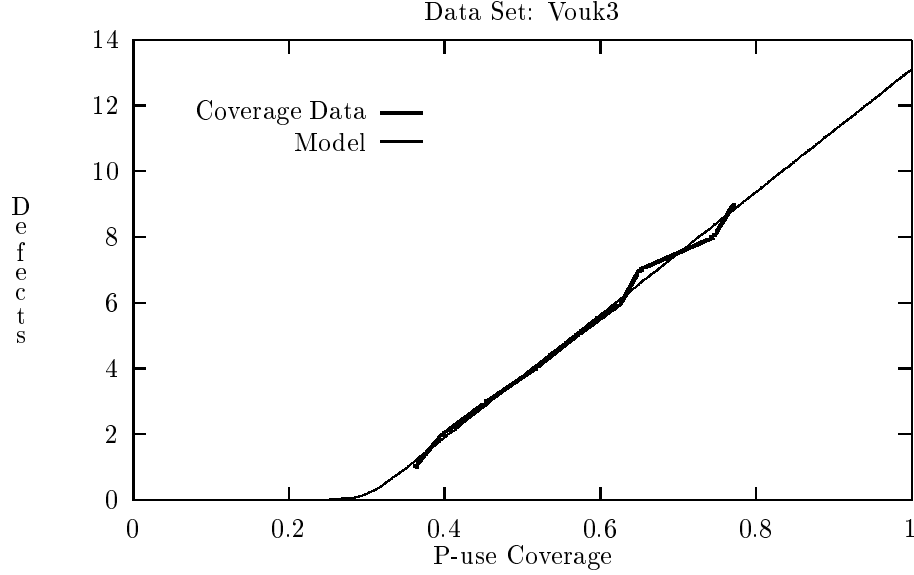


Figure 5.6: Defects vs. % P-use Coverage

Table 5.2: Projected number of total defects with 100% coverage (Vouk's data sets)

Data Set	Faults	Expected Faults		
	Found	Block	Branch	P-use
Vouk1	9	11	11	18
Vouk2	7	8	8	8
Vouk3	10	10	11	13

5.4 Significance of Parameters

We can obtain an analytical interpretation of the parameters of the coverage model by remembering that they are derived from similar parameters of the logarithmic model. From equation 5.4 we find that parameter a_0^i is equal to β_0^D/N_0^D . From chapter 3 we know that β_0^D is equal to $I_s D_{min}^D$ where I_s is the number of program source lines, and D_{min} is the defect density where the lowest possible fault exposure ratio occurs. Since N_0^D is the initial number of defects, we could restate this as the initial defect density D_0^D times the number of source lines I_s . This would then give us

$$a_0^i = \frac{B_0^D}{N_0^D} = \frac{I_s D_{min}^D}{I_s D_0^D} = \frac{D_{min}^D}{D_0^D}$$

From chapter 3 we know that we can estimate D_{min}^D as $D_0^D/3$ when D_0^D is greater than 10/KLOC and as three when it is less than this. Therefore the parameter a_0^i is a scale factor relating the enumerables covered to the defects covered. At high defect densities this scale factor is approximately one third, at low defect densities it rises quickly past one. This is consistent with the idea that unreliable programs will only have a small fraction of their defects uncovered by extensive testing, even though a large number of defects may be found. Conversely, extensive testing of reliable programs is likely to account for a large percentage of the total defects even if only a few are found.

Substituting equation 3.13 in to equation 5.5 we find that for parameter a_1^i

$$a_1^i = \frac{K_{min} e^{\frac{D_0^D}{D_{min}^D} - 1}}{K_{min}^i e^{\frac{D_0^i}{D_{min}^i}}} \quad (5.8)$$

Examining this equation it appears that the parameter a_1^i is related to the ratio between K_{min} and K_{min}^i ; that is the relationship between the minimum fault exposure ratio and the minimum enumerable coverage ratio. The ratio between overall fault exposure and enumerable exposure also appears to be a factor. Using the results presented in chapter 3 it may be possible to estimate the numerator of equation 5.8, but at this time we lack adequate data to make any statements about the values of D_{min}^i and K_{min}^i in the denominator. What we can say is that parameter a_1^i is dependent on the relationship between the fault exposure ratio and the enumerable coverage ratio. Stated another way, a_1^i is dependent upon how fast defects are found relative to how fast new enumerables are covered. It may be that this value can be reasonably estimated to be some constant, if we had more data.

Equation 5.6 gives the value of a_2^i as $D_0^i / I_s D_{min}^i$. It would be tempting to assume that the relationship between D_0^i and D_{min}^i is the same as between D_0^D and D_{min}^D , but at this point we have no justification for this. The most we can say at this point is that parameter a_2^i is based on the relationship between the enumerable density where enumerables are hardest to cover and the actual enumerable density, and scaled by the inverse of the number of source lines.

Taking what we know of these three parameters in to account, it appears that the most we can say about the coverage model at this time is that it depends on the defect density, the density of the enumerable used for modeling, and the ratio between how quickly each is covered.

5.4.1 A Linear Approximation

Figure 5.2, which is a direct plot of the model, and figures 5.3, 5.4, and 5.5 which plot actual data, suggest that at higher coverage values, a linear model can be a very good approximation. This leads us to an analytical interpretation of the behavior. This interpretation can be used for obtaining a simple preliminary model for planning purposes. We will obtain a linear model from equation 5.3. Let us assume that at higher values of C^i equation 5.3 can be simplified as

$$\begin{aligned} C^D(C^i) &= a_0^i \ln[a_1^i e^{a_2^i C^i}] \\ &= a_0^i \ln(a_1^i) + a_0^i a_1^i C^i \\ &= A_0^i + A_1^i C^i \text{ where } C^i > C_n^i \end{aligned} \quad (5.9)$$

$$A_0^i = a_0^i \ln(a_1^i) = \frac{\beta_0^D}{N_0^D} \ln\left(\frac{\beta_1^D}{\beta_1^i}\right) \quad (5.10)$$

and

$$A_1^i = a_0^i a_1^i = \frac{\beta_0^D}{N_0^D} \frac{N^i}{\beta_0^i}$$

Note that this simplification is applicable only for values of C_n^i greater than where the knee occurs. The values for the parameters of this model can not be obtained until a clear linear behavior beyond the knee has been established. Assuming that the knee occurs where the linear part of the model intersects the x-axis, using equation 5.9, the knee is at

$$C_k^i = -\frac{A_0^i}{A_1^i} \quad (5.11)$$

For a strict coverage measure, $C^D \approx 1$ when $C^i = 1$. Hence from equation 5.9 we have

$$A_0^i + A_1^i \approx 1.$$

Replacing A_0^i by $1 - A_1^i$ in equation 5.11, and using 5.10, we can write,

$$C_k^i = 1 - \frac{1}{a_0^i a_1^i}$$

This can be written as,

$$C_k^i = 1 - \left(\frac{D_{min}^i D_0^D}{D_{min}^D D_0^i} \right)$$

Where D_{min}^i , D_{min}^D , D_0^i are parameters and D_0^D is the initial defect density. Thus for lower defect densities, the knee occurs at higher test coverage. This has a simple physical interpretation. If a program has been previously tested resulting in a lower defect density, it is likely that the enumerables with higher testability have already been exercised. This means that testing will start finding a significant number of additional defects only after higher test coverage values are achieved.

5.5 Conclusions

Defect density is an important measure of software reliability, figuring prominently in the reliability assessment of many quality assurance engineers and managers. Existing methods for estimating the number of defects can underestimate the number of defects because of a bias towards easily testable faults. The exponential model tends to generate unstable projections and often yields a number equal to defects already found.

Experimental data suggests that our method can provide more accurate estimates and provide developers with the information they require to make an accurate assessment of software reliability. The choice of coverage measure does have an effect on the projections made, and we suggest that a strict coverage measure such as p-uses should be used for programs which must have very high reliability. We have presented the beginnings of an interpretation for the parameters of this model, but more work is required in order to refine this interpretation and allow estimates to be made based on the software development process.

Chapter 6

ROBUST : AN INTEGRATED SOFTWARE RELIABILITY TOOL

6.1 Introduction

In order to make it easier for industry practitioners to apply our research, we have developed a tool called ROBUST which aims to make using software reliability growth models easier. ROBUST also aims at integrating the results of our research, connecting the various aspects together to provide a coherent picture of project reliability. It does this using a graphical user interface which makes it easier to use than some previously available tools. Available for a wide variety of UNIX and Microsoft Windows platforms, we have distributed copies to various individuals in industry and academia. These users have contributed valuable feedback in program features and stability.

6.2 Capabilities

ROBUST provides a number of traditional software reliability growth models, and numerous means to assess their accuracy and increase it. An implementation of the static defect density model described in (Malaiya and Denton 1997) is available, and is integrated with the SRGM models in order to provide static models of software development based on quantitative measurements of the development process. ROBUST also provides for coverage based estimates of software reliability. It does all this using a simple graphic user interface which puts commonly needed techniques and information quickly within the users reach. Online help is provided, as well as an introductory tutorial and users manual (Denton 1997).

6.2.1 Traditional Software Reliability Models

ROBUST supports four different traditional software reliability growth models. All these models make use of either time to failure data or failure interval data. These two forms are essentially equivalent, and ROBUST can generate one form from the other, allowing all models to be used with either type of data without the user explicitly converting between forms.

Of the four models supported, the exponential and logarithmic models are the most popular and widely applicable. The rest of this work focuses extensively on these models. The power model is an old model, and not generally considered to be very accurate, but is included for the sake of completeness. The delayed-S shape model has been found to be useful for some software reliability data. For the first three models ROBUST can make estimates using either the least squares method or the maximum likelihood method. For the delayed-S model only the least squares models is supported due to the complexity of the maximum likelihood equations.

ROBUST defaults to providing a textual view of the data and the requested models. However, for ease of use it can also produce graphs of a model, overlaid with the data to which it is being applied. These graphs are available not only for the traditional time versus accumulated failures, but also for time versus failure intensity;

a view many tools do not provide. Figure 6.1 and 6.2 show views of the logarithmic model for the Musa T1 data set.

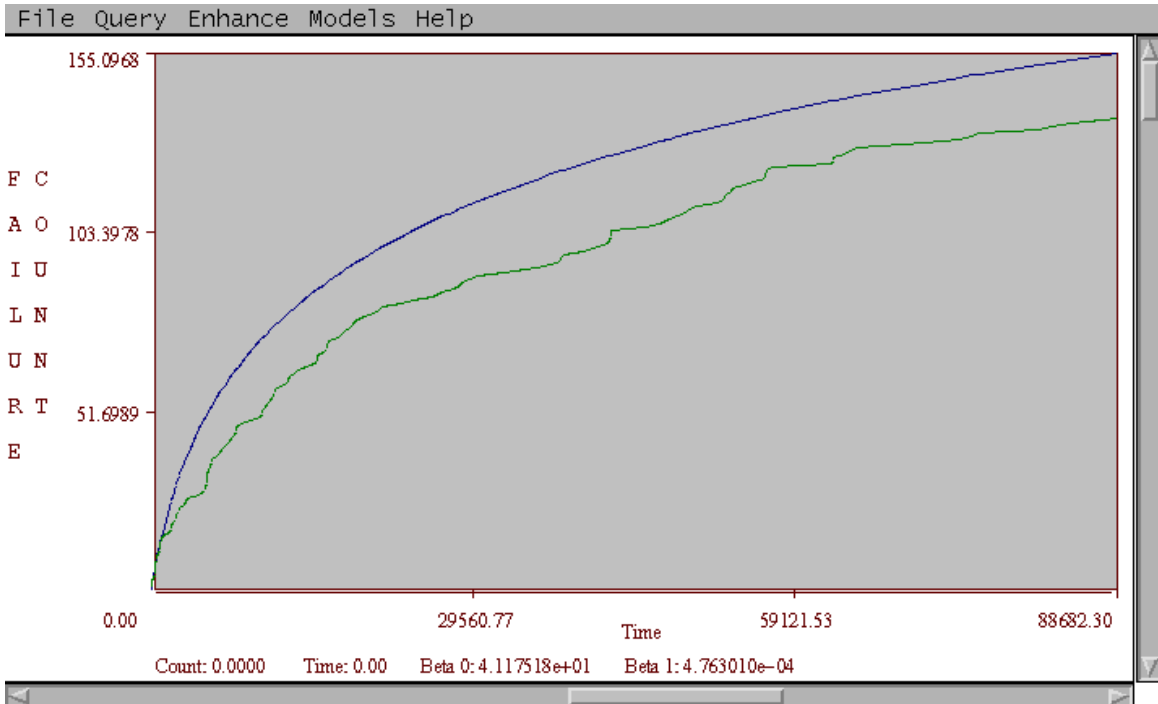


Figure 6.1: Cumulative failure count vs. time for T1 data set

Many tools support these models and estimation methods, so the contribution of ROBUST is not its ability to work with these models. Rather, ROBUST's contribution is in the techniques it has available to enhance the accuracy of these models. ROBUST implements both the fixed grouping and lump smoothing techniques suggested by Li, as well as his recalibration method (Li and Malaiya 1995). ROBUST also supports a stabilization feature which makes use of the weighted average technique discussed in chapter 4. This feature only works with the exponential and logarithmic models, but makes use of ROBUST's static defect density model to estimate the parameters for these models.

6.2.2 Static Defect Density Model

ROBUST supports the static defect density model put forward by Malaiya and Li in (Li and Malaiya 1995) and refined by Malaiya and Denton in (Malaiya and Denton 1997). A dialog box, shown in figure 6.3 allows the user to enter the parameters of the various sub models, and then obtain an estimate of final defect density. Additional fields allow the user to set values for the other parameters which go into making static estimates of model parameters as described in chapter 3. Using this dialog box the user can obtain estimates for these static parameters, and then use the modeling options on the main menu to produce static models based on their choices.

6.2.3 Coverage Model Support

The most unique ability of ROBUST is its support for coverage based reliability modeling. Coverage based reliability modeling removes the testing time element from models, and replaces it with the more objective measure of how many program enumerables, such as branches, are covered. ROBUST currently supports models based on blocks, branches, p-uses, and c-uses; using the model proposed by Malaiya et al. (Malaiya and Denton 1998a; Malaiya and Denton 1998b; Malaiya and Denton 1998c). Future versions will support arbitrary program enumerables as specified by the user.

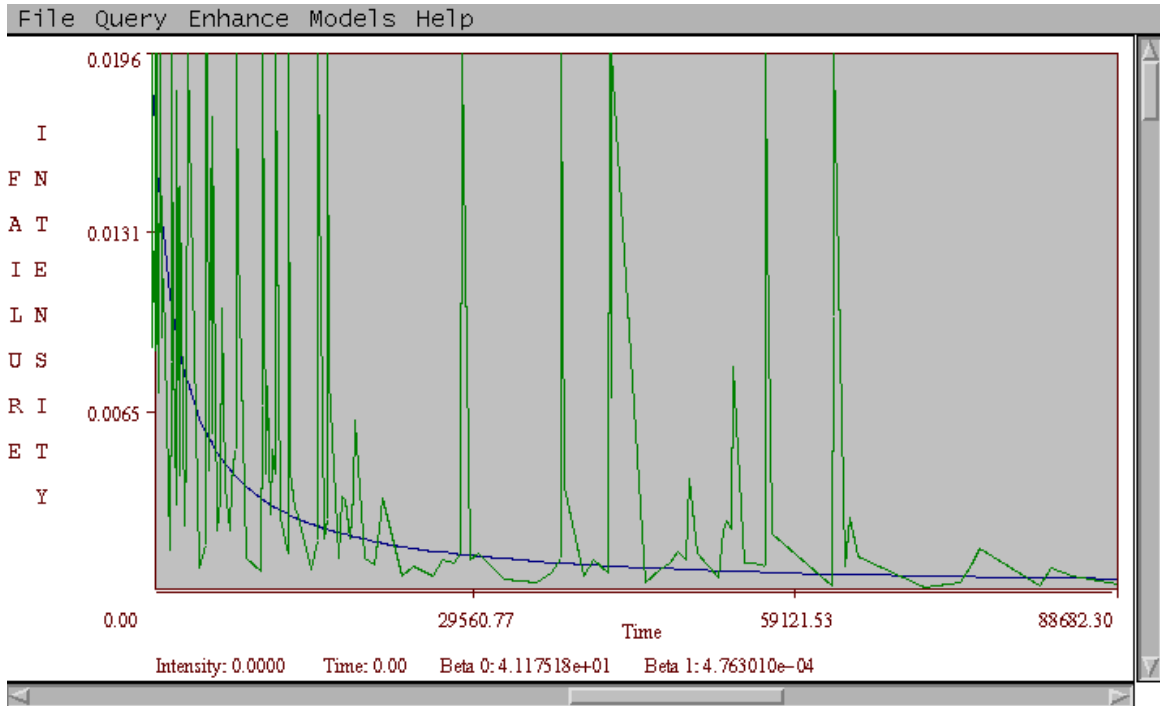


Figure 6.2: Failure intensity vs. time for T1 data set

Development Process

These parameters describe the development process

Constant: 10.00

Phase: System Testing

Dev. Team: Average

Maturity: Repeatable - 2

Structure: 10

Finished System

These parameters describe the characteristics of the finished system

CPU Rate: 4.00 Alpha: 10.00

Src/Obj: 4.00 Time: 50000.00

KLOC: 10.00 Direct Estimation

Model Parameters

Exponential Logarithmic

Beta 0: 109.1999960 Beta 0: 42.6824601

Beta 1: 0.0000194 Beta 1: 0.0000759

Hide Set

Figure 6.3: ROBUST's Static Defect Density Model Dialog

ROBUST provides not only a textual view of coverage data and a model, but also an interactive graphical display which allows the user to see how defect coverage grows with enumerable coverage, and select the point at which they expect testing to end; since 100% coverage of some enumerables is very difficult to obtain. Figure 6.4 shows ROBUST displaying this graph, with a user selected coverage level of 95%.

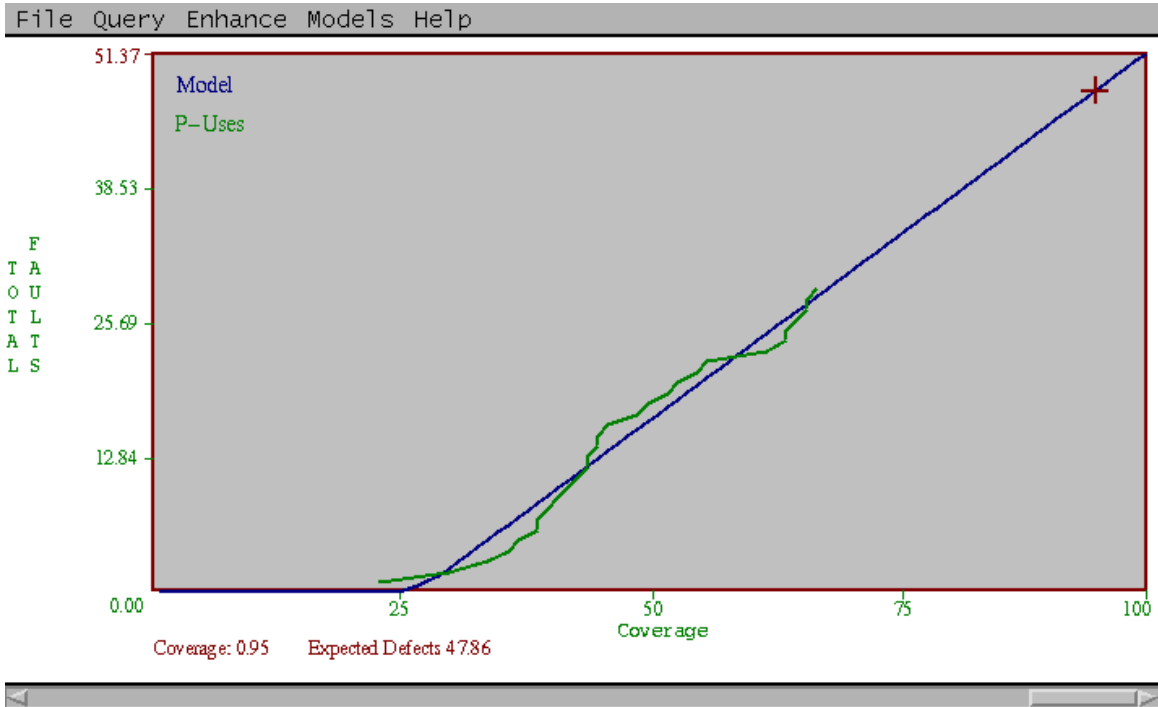


Figure 6.4: A Graphical View of ROBUST's Coverage Model

6.3 Development

ROBUST was developed with several goals in mind. First, it needed to be efficient. Fitting of models to data can be a computationally intensive process and our primary model accuracy measures require repeated fitting of a model, and we wanted the user to be able to obtain these measures quickly. This led us to choose to implement the final version in C and C++, instead of using Microsoft Excel and Visual Basic as done in the original prototype by Li (Li and Malaiya 1995).

Next, ROBUST needed to run on both Microsoft Windows and UNIX systems. This meant choosing a graphical toolkit that would enable us to migrate between platforms with little or no effort. We chose to use the freely available V toolkit (Wampler 1995). This provided us with all the features we needed, and allows us to compile the various UNIX versions of ROBUST and the Microsoft Windows version from the same set of sources.

We also felt that ROBUST would need a clean design, one that kept the numerics separate from the graphical implementation, and that allowed for new models and research results to be easily incorporated into the program. This design goal was obtained by implementing high level model generation and accuracy routines independent of the supported models, model specific behavior was then implemented using pointers to the relevant functions. Keeping the numerics code separate from the user interface allowed us to easily develop the batch programs which generated most of the results in this thesis and have aided in much of our other research.

Finally, ROBUST needed to live up to its name and be largely bug free. Testing against previously existing tools with similar functions, and hand checking of results verified the accuracy of the numerics routines, while extensive testing over the last three years has removed many of the bugs present in the user interface. Although new bugs are discovered from time to time, there are currently no known bugs in ROBUST.

6.3.1 A Java Version

When development on ROBUST first began, the Java language and environment from Sun had only recently become available. The Java system was designed to allow “write once, run anywhere” functionality for GUI based applications and web browser based applets, in a reasonably efficient manner. In keeping with our original design goal of a tool that would work on either a Microsoft Windows platform or various UNIX variants, Java offered a cross platform GUI option. At the time however, it was felt the Java was not yet mature enough to fully support all of our design goals, particularly in terms of efficiency.

Over the course of the project, this has changed. Java is now a robust and efficient platform, with an advanced GUI toolkit capable of meeting all our requirements. Its acceptance has grown, and some version of Java now ships with or is available separately for all modern computers and operating systems. A new version of ROBUST written in Java is now under development, primarily by Brian McGuire. This version possesses an easier to use interface, with improved reporting capabilities. It is capable of running as a stand alone application or from within a browser, making it ideal for deployment over the world wide web. Careful attention to model implementation and the underlying numerical code allows for the various portions of the program to be more easily reused than the previous versions the ROBUST numerics library, thanks to the object oriented nature of Java. In the future, it is likely that this will be the most commonly used version of ROBUST.

6.4 Comparison to Other Tools

Several programs besides ROBUST exist to facilitate the use of software reliability models. The two most notable are the text based SMERFS and its graphical cousin CASRE. Both programs provide a large range of traditional software reliability models, and CASRE provides some additional enhancement tools. Neither package implements recalibration or the type of smoothing that ROBUST employs, and both packages are strictly limited to working with traditional models; neither has the capability of producing a static model or of working with coverage based data.

6.4.1 SMERFS

One of the oldest software reliability tools, and one that is still being used in many places, is SMERFS by Bill Farr (Farr and Smith 1988). SMERFS provides a large number of models, including all those that ROBUST supports and several more, most notably the Littlewood-Verrall model in both its linear and inverse polynomial forms. SMERFS offers least squares and maximum likelihood estimation where appropriate, and provides a number of data conversion modules to make importing data easy, including features to convert from wall clock time to execution time, and vice versa. For each model supported SMERFS provides parameter estimates, current failure intensity at the time testing stopped, predictions of future faults, and other useful information dependent on the model selected.

SMERFS is a complete and comprehensive package, with the ability to produce either text based graphs, or output suitable for external plotting programs. However, SMERFS is solely a text based package. Although its option screens are clear and easy to follow, it has no way to produce its own highly accurate graphs. SMERFS’s text based operation makes it somewhat harder to use than is strictly necessary given the prevalence of GUI’s in today’s modern computing environment.

6.4.2 CASRE

CASRE addresses SMERFS’s most serious shortcoming, the lack of a graphical user interface. Available on both UNIX and Microsoft Windows platforms, CASRE makes use of the same FORTRAN code which drives SMERFS’s number crunching abilities. It provides graphical display abilities for both data and models, and incorporates several other capabilities not found in SMERFS. CASRE is the result of research by its author Allen Nikora and several others, and the most significant ability found in CASRE but not in SMERFS is the ability to create new models that are a combination of previous models, as described in (Lyu and Nikora 1992a). CASRE also has several data smoothing techniques, which are based on the assignment of different weights to different data points.

6.5 The Future of ROBUST

ROBUST has now reached the point where it offers the core features demanded by most software reliability engineering practitioners while being reasonably free of bugs. It supports most of the major SRGMs in use today, and it would not be difficult to add more; the JAVA version of the project promises to make it easy for future developers to add new models without ever having to look at the original source code. Both versions of ROBUST incorporate our research to date, and in the future it is expected that both will continue to incorporate new research into the integration between static parameter estimates and dynamic models, as well as ongoing research into coverage based models.

Chapter 7

CONCLUSION

We have examined the exponential and logarithmic models in depth, presenting summaries of previous work done to improve them. We have built on this by examining the issue of how parameters for these models are estimated. Although the existing body of literature spends a great deal of time discussing the derivation of the maximum likelihood equations for estimating the parameters of these models, it provides little practical advice to practitioners on which algorithms should be employed to solve these equations. Discussion with practitioners has suggested that in many cases they have dealt with this issue by linearizing the basic model form and fitting a straight line to the data, although this approach does not appear in the literature. We present the linear forms of the equations used to do least squares fitting here, and then make some recommendations about how to solve the maximum likelihood equations. We compare these two methods not only on raw data, but on data which had been smoothed following advice from Li (Li and Malaiya 1995). Our results show that for the exponential model the least squares fitting technique does not provide good predictive capabilities, and suffers from a general inability to find valid parameter estimates; for this model the maximum likelihood method is generally superior. When smoothing is applied to the data the least squares method is somewhat more promising, although there is still the problem of inaccurate parameter estimates in many cases. Our results with the logarithmic model are similar, but suggest that for some data the least squares technique might be able to obtain parameter estimates for the logarithmic model which are as good or better than the ones obtained from the maximum likelihood technique, at least in terms of long term model accuracy.

In both cases the maximum likelihood method requires the use of iterative fitting routines, which may be subject to local minima and thus may not produce the best possible parameter estimates. This issue may be worth exploring, examining the space in which the routine searches for parameter estimates and assessing the effects of the initial parameter choice on the final results. It may be that using the least squares estimates as a starting point for the maximum likelihood technique proves useful, in effect using MLE to tweak the results of the simpler least squares method.

In chapter 3 we discussed the meaning of the parameters of the exponential model, and presented an interpretation for the parameters of the logarithmic model. We showed that if the logarithmic model holds, then the fault exposure rate is variable. We presented a means of estimating the parameters of the exponential model, and two means of estimating the parameters of the logarithmic model, all based on the development process. This allows us to make predictions based on the models prior to the start of testing. Although we do not expect such predictions to be superior to ones made using data gathered from testing, early test data can be unstable and lead to erroneous predictions. In some cases our static estimates based on our understanding of the model to be used and the development process might be superior to those based on early test data.

To overcome the problems inherent in working with early failure data, without sacrificing the information present in late term failure data we proposed a scheme called stabilization which combines statically estimated parameters with ones obtained from fitting the model to the data set. For the exponential model it appears that it is best to use the static parameters to set acceptable bounds on the dynamically fitted parameters. When the dynamic parameters exceed this range we take it as an indication that the model has failed to conform to the data, and replace the fitted parameter β_0 with the static estimate before using the model. For the logarithmic model we find that it is better to use a weighted average of the two parameter estimates, favoring the dynamically fitted parameters with 75% of the weight. This technique works particularly well when the data

in question has been subjected to two rounds of lump smoothing.

Although stabilization in general appears to be useful, there are still concerns. The usefulness of stabilization depends heavily on our ability to estimate the initial number of defects, and although there are many tools for doing so it still remains an important task. When stabilization is used, the practitioner must be careful with their estimates of initial defect density. If the direct method of parameter estimation is used to set the static parameters of the logarithmic model, then accurate estimation of initial defect density is not so critical, however our results suggest that this method has its own problems; attempts to use the direct method of parameter estimation for stabilization of the logarithmic model generally failed. This suggests that we must continue to refine our understanding of the parameters of the logarithmic model. In particular, the relationship between defect density and fault exposure ratio should be examined closely because it provides the basis for most of the important terms in the equations 3.12 and 3.13 which define the parameters of the logarithmic model.

Traditional software reliability models suffer from varying test effectiveness, an hour of testing using one test suite may not be as effective as a lesser amount of time spent using a different test suite. Issues such as program testability and processor speeds complicate this issue even further. To address these issues Malaiya et al. (Malaiya, Li, Bieman, Karcich, and Skibbe 1994; Malaiya and Denton 1998b) have developed a software reliability model based on objective measures of program coverage, such as the percentage of blocks or branches covered. This model predicts the number of defects found as a function of coverage, and is thus not vulnerable to issues such as varying test effectiveness. We present the latest work on this subject, and elaborate on the meaning of the parameters of this model. The main controlling factor in this model appears to be the relationship between defect density and the density of the program enumerable that the user chooses to base the model on. In the future we would expect to be able to make estimates about the values of the parameters of this model based on our understanding of them and the development process used to create the project in question. Unfortunately, at this time we lack the data necessary to make such recommendations. More work is needed to assess the enumerable densities of various types of programs, and how quickly these enumerables are covered by test cases.

Over the course of this work we have developed a program called ROBUST. This integrated software reliability tool implements our findings in a manner which makes it easy for software reliability practitioners to use. It provides a number of traditional software reliability growth models, and several enhancement techniques to improve the reliability of these models. It integrates our understanding of the parameters of the logarithmic and exponential models into the static defect density model presented by Malaiya and Denton in (Malaiya and Denton 1997) to allow for stabilization as presented in chapter 4 and for models based completely on static parameter estimates. It provides the first implementation of the Malaiya et al. coverage model. It does all this using an easy to use GUI interface, and is available on a variety of UNIX and Microsoft Windows platforms. Future development on this tool and a Java based version will expand the number of available models and further integrate its coverage based modeling abilities with its ability to make predictions based on the software development process.

We have examined software reliability growth models, the difficulties in their use and the techniques which can make them more reliable. We have shown that an understanding of their parameters can lead to more accurate long term predictions from such models, and go on to propose a new type of software reliability growth model that addresses many of the fundamental shortcomings of traditional models. More work is needed to further understand how the software development process influences the parameters of this model.

REFERENCES

- Abdel-Ghaly, A. A., P. Y. Chan, and B. Littlewood (1986, September). Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering SE-12*(9), 950–967.
- Binder, R. V. (1997). Six sigma: Hardware si, software no! <http://www.rbsc.com/pages/sixsig.html>.
- Brocklehurst, S., P. Y. Chan, B. Littlewood, and J. Snell (1990, April). Recalibrating software reliability models. *IEEE Transactions on Software Engineering 16*(4), 458–469.
- Brocklehurst, S. and B. Littlewood (1992, July). New ways to get accurate reliability measures. *IEEE Software 9*(4), 34–42.
- Brocklehurst, S. and B. Littlewood (1996). Techniques for prediction analysis and recalibration. In M. Lyu (Ed.), *Handbook of Software Reliability Engineering*, Chapter 4, pp. 119–166. New York: IEEE Computer Society Press.
- Butler, R. W. and G. B. Finelli (1993, January). The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering 19*(1), 3–12.
- Chen, M., M. R. Lyu, and W. E. Wong (1996, March). An empirical study of the correlation between coverage and reliability estimation. In *Proc. IEEE Third International Symposium on Software Metrics*, Berlin, Germany, pp. 133–141.
- Denton, J. A. (1997, June). *ROBUST, An Integrated Software Reliability Tool*. Colorado State University.
- Farr, W. (1996). Software reliability modeling survey. In M. Lyu (Ed.), *Handbook of Software Reliability Engineering*, Chapter 3, pp. 71–118. New York: IEEE Computer Society Press.
- Farr, W. and O. Smith (1988). Smerfs user’s guide. Technical Report TR 84-373, Naval Surface Warfare Center.
- Follenweider, R., R. Karcich, and G. J. Knafl (1993, November). A systematic approach to software reliability modeling. In *Proc. Fourth International Symposium of Software Reliability Engineering*, Denver, pp. 62–70.
- Frankl, P. and E. Weyuker (1988, October). An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering 14*(10), 1483–1498.
- Goel, A. L. and K. Okumoto (1979, August). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transaction on Reliability RE-28*(3), 206–210.
- Hatton, L. (1997, Nov./Dec.). N-version design versus one good design. *IEEE Software 14*(8), 71–76.
- Hutchings, M., T. Goradia, and T. Ostrand (1994). Experiments on the effectiveness of data-flow and control-flow based test data adequacy criteria. In *Proc. International Conference on Software Engineering*, pp. 191–200.
- Jelinski, Z. and P. Moranda (1971). Software reliability research. In W. Freiberger (Ed.), *Statistical Computer Performance Evaluation*, pp. 465–484. Providence, RH: Academic Press.
- Kanoun, K. and J. Laprie (1996). Trend analysis. In M. Lyu (Ed.), *Handbook of Software Reliability Engineering*, Chapter 10, pp. 401–437. New York: IEEE Computer Society Press.

- Lab, R. (1992). Methodology for software reliability prediction and assessment. Technical Report RL-TR-95-52, Vol. 1 and 2, Rome Labs.
- Lahey, P. and A. Neufelder (1997). *System and Software Reliability Assurance Notebook*. Rome, New York: Rome Laboratory.
- Li, N. (1996). *Measurement and Enhancement of Software Reliability Through Testing*. Ph. D. thesis, Colorado State University.
- Li, N. and Y. K. Malaiya (1993, November). Enhancing accuracy of software reliability prediction. In *4th International Symposium on Software Reliability Engineering*, Denver, pp. 71–79.
- Li, N. and Y. K. Malaiya (1995, October). Robust: A next generation software reliability engineering tool. In *Proc. International Symposium on Software Reliability Engineering*, France.
- Li, N. and Y. K. Malaiya (1996). Fault exposure ratio: Estimation and applications. In *Proc. of the IEEE International Symposium on Software Reliability Engineering*, pp. 372–381.
- Lyu, M. and A. Nikora (1992a, July). Casre - a computer-aided software reliability estimation tool. In *Proc. of the Fifth International Workshop on Computer-Aided Software Engineering*, Montreal, Que., pp. 264–275. IEEE Computer Society.
- Lyu, M. R., J. R. Horgan, and S. London (1993). A coverage analysis tool for the effectiveness of software testing. In *Proc. of the IEEE International Symposium on Software Reliability Engineering*, pp. 25–34.
- Lyu, M. R. and A. Nikora (1992b, July). Applying reliability models more effectively. *IEEE Software* 9(7), 43–52.
- Malaiya, Y. and J. A. Denton (1998a, November). Estimating the number of defects: A simple and intuitive approach. In *International Symposium of Software Reliability Engineering*, Paderborn, Germany.
- Malaiya, Y. K. (1991, May). Early characterization of the defect removal process. In *Proc. Ninth Annual Software Reliability Symposium*, pp. 6.1–6.4.
- Malaiya, Y. K. and J. A. Denton (1997, November). What do software reliability parameters represent? In *Proc. International Symposium on Software Reliability Engineering*, Albuquerque, NM, pp. 124–135.
- Malaiya, Y. K. and J. A. Denton (1998b). Estimating defect density using test coverage. Technical Report 98-104, Colorado State University, Ft. Collins, CO.
- Malaiya, Y. K. and J. A. Denton (1998c, November). Estimating the number of residual defects. In *Third IEEE International High-Assurance Systems Engineering Symposium*, Washington D. C., pp. 98–195. IEEE.
- Malaiya, Y. K., N. Karunanithi, and P. Verma (1992, December). Predictability of software-reliability models. *IEEE Transactions on Reliability* 41(4), 539–546.
- Malaiya, Y. K., N. Li, J. Bieman, R. Karcich, and B. Skibbe (1994, November). The relationship between test coverage and reliability. In *Proc. of the International Symposium on Software Reliability Engineering*, pp. 186–195.
- Malaiya, Y. K., A. von Mayrhauser, and P. Srimani (1993, November). An examination of fault exposure ratio. *IEEE Transactions on Software Engineering* 19(11), 1087–1094.
- Malaiya, Y. K. and S. Yang (1984, October). The coverage problem for random testing. In *Proc. IEEE International Test Conference*, pp. 237–245.
- McConnell, S. (1997, May/June). Gauging software readiness with defect tracking. *IEEE Software* 14(3), 135–136.
- Musa, J. D. (1975). A theory of software reliability and its application. *IEEE Transactions on Software Engineering* SE-1(3), 312–327.
- Musa, J. D. (1979). Software reliability data. Technical report, Rome Air Development Center, Griffis Air Force Base, N. Y.
- Musa, J. D., A. Iannino, and K. Okumoto (1987). *Software Reliability - Measurement, Prediction, Application*. New York: McGraw-Hill.

- Musa, J. D. and K. Okumoto (1984). A logarithmic poisson execution time model for software reliability measurement. In *Proc. Seventh International Conference on Software Engineering*, Orlando, Florida, pp. 230–238.
- Pasquini, A., A. N. Crespo, and P. Matrella (1996, December). Sensitivity of reliability growth models to operational profile errors. *IEEE Transactions on Reliability* 45(4), 531–540.
- Piwowarski, P., M. Ohba, and J. Caruso (1993, May). Coverage measurement experience during function test. In *Proc. of the 15th International Conference on Software Engineering*, pp. 287–300.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1988). *Numerical Recipes in C* (2nd ed.). Cambridge University Press.
- Schneidewind, N. F. (1993, November). Software reliability model with optimal selection of failure data. *IEEE Transactions on Software Engineering* 19(11), 1095–1104.
- Srivastava, A. (1992). Unreachable procedures in object-oriented programming. *LOPLAS I*(4), 355–364.
- Vouk, M. A. (1992, October). Using reliability models during testing with non-operational profiles. In *Proc. of the 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation*, pp. 103–111.
- Wampler, B. (1995). V : A c++ gui framework. <http://www.objectcentral.com>.