49
33

A Reliability Model Incorporating Software Quality Metrics

by

Ashok Yerneni

Thesis Submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED :

Dr. Kafura, Dennis Chairman

Dr. Arthur, James D.

Dr. Henry, Sallie

September, 1989
Blacksburg, Virginia

A Reliability Model Incorporating Software Quality Metrics

by

Ashok Yerneni

ABSTRACT

Large scale software development efforts in the past decade have posed a problem in terms of the reliability of the software. The size and complexity of software that is being developed is growing rapidly and integrating diverse pieces of software in the operational environment also poses severe reliability issues, resulting in increased development and operational costs. A number of reliability models have been defined in the literature to deal with problems of this kind. However, most of these models treat the system as a "black box" and donot consider the complexity of the software in its reliability predictions. Also, reliability is predicted after the system had been completely developed leaving little scope for any major design changes to improve system reliability.

This thesis reports on an effort to develop a reliability model based on complexity metrics which characterize a software system and runtime metrics which reflect the degree of testing of the system. A complete development of the reliability model is presented here. The model is simple and reflects on our intuition of the software development process and our understanding of the significance of the complexity metrics. Credibility analysis is done on the model by simulating a number of systems and applying the model. Data collected from three FORTRAN coded systems developed for NASA Goddard was used as representative of the actual software systems. An analysis of the results is finally presented.

ACKNOWLEDGMENTS

I wish to express my deep gratitude to Dr. Dennis Kafura, without whose patient guidance and invaluable technical advice this work would not have been possible. I also wish to thank Dr. Sallie Henry and the project team for their valuable comments during the model development phase. I would also like to thank Dr. James D. Arthur for serving on my committee.

Table of Contents

List of Tables

List of Figures

Chapter 1

1.0 Introduction

1.1 Motivation for the Study

Software design and development is the key to effective use of today's increasingly faster hardware technology. However large scale software development efforts have put forth a problem of another kind : reliability of software systems. The size and complexity of software is dramatically increasing, and with networking and multiprocessing, many diverse pieces of software are simultaneously running and interacting with each other. This slows down the software development process and increases the development and operational costs. Also, the past decade has seen a proliferation of personal computers, workstations, mini's, micro's and mainframes; resulting in heavy dependence on these systems. Breakdown of these systems could be expensive and even catastrophic.

Software design and development is still considered an 'art' and there is a severe shortage of skilled programmers who can do the job 'right'. There are as yet few automated techniques for generating software. Rapid evolution of computer technology is also creating its own problems. Software tends to become economically obsolete more rapidly and one cannot spend enough resources and time to create the highest quality software products and still be economically viable. Thus most often, because of cost and schedule pressures, tradeoffs have to be made between high quality, rapid delivery, and low cost[MUSA87]. This means that any method of reliability assessment and/or improvement of these software systems could be potentially valuable.

Currently, there are two methodologies in developing reliable software for *large* systems[LEW88] :

1) Divide and conquer approach. This is the structured programming approach of breaking up a large problem into manageable subproblems with well defined interactions. It is a widely used technique and serves to reduce the complexity of tasks a designer has to handle at a time.

2) Use of software fault tolerant techniques such as the n-version programming[AVIZ77] and recovery blocks[RAND75]. However, this approach leads to increased size and complexity of the software and also it has not been demonstrated that these techniques can actually improve reliability.

A third approach is the concept of 'reusability' in the programming environment. This method involves creating libraries of useful modules either using conventional programming styles or the relatively new and as yet unproven, object oriented programming paradigm. This technique has scope to reuse software, reduce development time and possibly decrease software costs. Most software developers reuse software to some extent. However, large software is typically developed in a team environment with distribution of tasks. Therefore, pieces of software tend to be written in different styles. Consequently, errors get introduced during the integration phase when modifications need to be made to get the software system to work as a unit. To err is human, and so no design approach can guarantee perfectly reliable software. Therefore, there is an even greater need for reliability measures that predict software system failure.

Other approaches that exist and could be used to produce reliable software but which might not be tangible for large systems are: exhaustive testing of software using automated test generation schemes and using formal proof of correctness.

The purpose of this study is to explore a software reliability model which attempts to predict software system failure. The model reflects our intuitive understanding of the software development methodology and the human aspects of the software development process. The modelling employs software complexity metrics which are based on the the structure of the code and runtime metrics which reflect the degree of testing to which the system and its components have been subjected. The model does not assume any specific complexity metric. Thus, any software metric which is known to correlate with errors can be used in the model.

Credibility analysis is performed on the reliability model to determine if the model performs to our expectations. For the analysis, we have simulated a number of system descriptions which are representative of the variance in properties of the actual software systems. We used the data compiled from the three Fortran coded systems developed for NASA Goddard[NASA82] and generated system descriptions as a variation of the individual properties associated with it.

1.2 Survey of existing reliability models

The literature cites a large number of models for estimating software system reliability. Most of these models are based on the execution behavior observed during system testing. Referred to as the 'black box' approach, this approach suffers from two limitations. First, software system design is heavily influenced by it's applications. Thus design approaches tend to differ significantly between systems. Also, code development in a project environment is also influenced by coding standards adopted by the group. The 'black box' approaches do not consider these differences in modelling software reliability. Software metrics capture these differences to some extent and are increasingly gaining confidence as measures of software reliability. Secondly, the execution behavior of the system is available

only when the system is completely built. Thus, the 'black box' reliability models have applicability only when the system has been completely built and there is little scope for improving system reliability.

Most 'black box' models assume a hazard function (failure density) and determine the model parameters using Maximum Likelihood Estimators. An early model by Schick and Wolverton[SHICK73] assumes that error rate (hazard rate) is proportional to the number of errors remaining and the amount of time spent in debugging, and computes a reliability estimate and an estimate of the total time required to find all the remaining errors. A modified model[SHICK78] assumes that hazard rate is a parabolic function. Moranda[Morand75] presented two models which assume that the hazard rate between successive errors decreases in steps that form a geometric progression. Both compute a reliability estimate and mean time to failure (MTTF). Sukert[SUKERT77] developed a reliability growth model from data gathered from a sequence of test stages. The model does not present any criteria for fixing the number of tests per stage and the number of stages in the test-sequence. It predicts the reliability at each stage and uses least squares approximation to determine the parameters. Shooman[SHOOM72] presented a model which assumes that the time-to-failure has an exponential distribution and the code size remains constant. The model determines a reliability estimate and the number of errors remaining in the system. Other 'black box' models include the Goel and Okumoto Bayesian model[GOEL78], Weibull model[WAGON73], Littlewood and Verall Bayesian model[LITTLE74], Shooman and Trivedi Markov model[SHOOM76].

A notable exception to the "black box" reliability models cited in literature is the quantitative model developed by Shooman. The model uses structural information from the system to determine the system failure rate and the system probability of failure[SHOOM76]. He uses metrics such as the frequency with which each of the paths are run (f_j), the running time for each path (t_j) and the probability of

error along each path (q_j) in the model. The term "path" was used in a general sense to mean any module, function, mode, etc. The model was developed under the assumption that the program was designed in a structured and modular fashion and the paths in the program are independent of each other. From the above information, Shooman estimated the times for successful and unsuccessful traversal of each path over N test cases and then arrived at the following system failure rate and system probability of failure :

(note : i refers to the total number of paths in the system and j is an instance of i)

$$\text{System Failure Rate} = \frac{\sum_{j=1}^{i} f_j * q_j}{\sum_{j=1}^{i} f_j * (1 - \frac{q_j}{2}) * t_j}$$

$$\text{System Probability of Failure} = \sum_{j=1}^{i} f_j * q_j$$

Along the same lines, we investigated a reliability model that incorporates information about the quality of the system as measured by software complexity metrics. In addition, the model we developed employs runtime metrics such as the amount of testing each component has undergone so far, and the number of errors discovered. The term "component" in our interpretation is defined as a procedure in the software system although any other basic unit (e.g. object, module) could be used as well. Since, metrics begin to appear during the development phase of the software system, an assessment could be made about the quality and reliability of the system during the development phase itself. Before proceeding further into the development process, decisions based on complexities could also be

made on the degree of testing that should be done to establish desired confidence levels on the reliability of the system.

Most models in the literature, determine reliability in the time domain (either calender time or execution time). That is, they determine quantities like the Mean Time To Failure [MTTF], Mean Time Between Failures[MTBF], or Failure Intensity (failures experienced in a time interval). In our model development however, we determine the probability of failure of the system in the next run. No assumptions are made regarding the execution time of each test case. The domain here is the number of test cases and no execution time projections are made. This makes it harder to make comparisions between the model and other existing reliability models. It is hoped that execution time complexity metrics when incorporated into the model, would transform the model into the time domain and facilitate the comparison of the model performance with the other reliability models. We have performed the credibility analysis of the model to determine if the model performs to our expectations and is directionally correct.

Chapter 2 presents the model development, the assumptions made in deriving the model and their justifications. Chapter 3 presents the software system simulator and the theory behind the simulation analysis. Chapter 4 presents a credibility analysis of the model with a discussion of the results. Chapter 5 concludes with summary of the model performance and future directions.

Chapter 2

2.0 The Reliability Model

This chapter presents the reliability model. Section 2.1 outlines the intuition behind the model and lists and justifies the assumptions made in the model development. The assumptions reflect the attributes of complexity metrics, the testing process, and the software system design and development process. Section 2.2 defines the reliability model.

2.1 Model Development

The reliability model being defined derives information from two sources: system design characteristics and system testing. The system design characteristics are represented by the software complexity metrics and the number of components in the system. As mentioned earlier, "component" is a procedure in the software system and each component in the system is associated with a value measuring it's "complexity". Numerous complexity measures have been defined in the literature (e.g., [MCCL78], [KAFU81a], [MCCA76]). The model development here assumes that the complexity metrics (kind of programming constructs used, number of variables, structure of code, interface between components, etc.) are valid measures of the program complexity and does not specify any particular metric. System testing is described by the number of test cases performed at each point, the number of failures observed in each component during these test cases, and the number of times a component was executed.

A number of assumptions have been made which reflect our intuition of the software development process, the nature of the complexity metrics, and the software testing process.

Assumption 1 : The probability of discovering an error in a component is inversely proportional to the degree of testing to which the component has been subjected without failure.

This assumption expresses the belief that the reliability of a component increases as the component executes without failure on an increasing number of test cases.

Assumption 2 : The probability of discovering an error in a component is directly proportional to the complexity of the component.

This assumption implies that there is a correlation between the component complexity and errors.

Assumptions 1 and 2 are the basis for the model development and identify our intuition of the system behavior and the assumptions about the complexity metrics. In addition, other simplifying assumptions have been made and are presented later in this chapter.

The model assigns a probability of failure to each of the components in the system based on the complexity metrics. One would be very skeptical about the reliability of a component with a very high complexity value. It indicates code in which we would expect a large number of errors to be found. Therefore, high complexity components initially have a high probability of failure. As testing progresses, if the component is executed without failure, confidence in the component grows and, hence, the probability of failure of the component drops. The rate at which the probability of failure drops is initially small, and grows faster with increasing number of test case executions. The probability of failure of the component ultimately stabilizes around a small value.

On the other hand, if the component complexity is low, it means that the code is small, simple and/or well written. Therefore, we do not expect many errors in such components; these components would initially have a low probability of failure relative to the high complexity components, and this probability drops very fast with increasing testing. The probability of failure of the component ultimately stabilizes around a small value.

Apart from the two assumptions already given, there are other underlying assumptions which are not directly visible and relate to testing, error discovery and error correction :

Assumption 3 : All failures which occur are observed. A failure is anything that produces results other than those expected.

Assumption 4 : Each test case results in zero or one failures. As soon as we discover a failure, the test case is deemed to have ended.

Assumptions 3 and 4 are self-explanatory and relate to error discovery. Strictly speaking, these assumptions are not true. However, they serve to simplify the model development without sacrificing the validity of the model and can be removed at a later stage.

Assumption 5 : Test cases are independent of each other.

Assumption 6 : Each test case is directed at testing a different aspect of the system.

Assumptions 5 and 6 relate to system testing. Errors get introduced into the system as a random phenomena resulting from human mistakes. Error discovery as a result of testing (unless selection of testing strategy is influenced by the errors discovered earlier) also tends to be random. Therefore, the chances of one error influencing

the other during error discovery is small [Musa79]. Together, assumptions 5 and 6 require that the testing process should be designed such that the testing sequence is large and varied enough to cover the entire system. System probability is determined from the components which get tested in the system. Thus if assumption 6 is not true, there is a possibility that, if a significant portion of the system remains untested initially, the predicted system probability of failure could be misleadingly low. Subseqently, if the untested portion gets tested and we begin to discover errors, the system probability of failure will not be very sensitive due the dampening effect on the probability from components which have been thoroughly tested.

Assumption 7: Every error discovered is corrected before the next test case is executed.

Assumption 8 : Correcting an error does not introduce additional errors and also does not change the complexities of the existing components.

Assumptions 7 and 8 relate to the error correction process. Strictly speaking, these assumptions are not true. There is always a chance that the error could get masked for this test case after the corrections and reappear at a later stage under different circumstances. Also, correcting an error could introduce additional errors and/or complexity because the correction process involves adding/modifying code. However, these assumptions simplify the model development and could be discarded at a later stage.

Assumption 9 : The change in probability of failure of a component with additional testing follows the standard normal cumulative distribution function.

The cumulative normal distribution curve has the properties which relate to our intuition of the component behavior. Moreover,

the normal curve is asymptotic on both ends and, hence, appropriate for our requirements. More will be said later about this assumption.

With the analysis presented above, we develop below a probability model for determining the system probability of failure.

2.2 Probability model

This section presents a complete development of the reliability model. Section 2.2.1 introduces the notation used in the model development. Section 2.2.2 discusses the assignment of initial probability of component failures and the progression of the component probabilities with testing. Section 2.2.3 discusses error discovery and corrections to the component probabilities on error discovery. In Section 2.2.4, we derive the system probability of failure.

2.2.1 Terminology :

The notation used in the model development is presented below and is self-explanatory.

T	Total number of components in the system.
N	Total number of test cases.
i	index over test cases : $1 \leq i \leq N$.
j	Index over components in the system : $1 \leq j \leq T$
c_j	Complexity of the jth component.
e_{ji}	Total number of errors discovered in component j after test case i.
n_{ji}	Number of test cases in which component j was executed after test case i.
p_{ji}	Probability of failure of component j after test

case i.

P_i System probability of failure after test case i.

However, in the subsequent treatment, we will not use the subscript i when it is clear from the context (i.e., we would simply write e_j , n_j, P_j, P_s instead of e_{ji}, n_{ji}, P_{ji}, P_{si} where ever it is convenient).

2.2.2 Assignment of Initial Probabilities

The initial probability of component failures are assigned based on the the component complexity values as measured by a software complexity metric. As stated in assumption 9, we use the cumulative normal distribution function for the initial probability of failure assignment to the components based on their complexities and subseqently, for tracking the component probability of failure with testing. The range of probabilities are linearly distributed over a specified **RANGE** of the curve, with the mean complexity component having an initial probability of failure which is 0.5*SCALE. The **SCALE** factor ranges from 0 to 2 and has the effect of shifting the **RANGE** of the probability assignment on the 0-1.0 scale. Figure 1 illustrates this assignment. The quantities **RANGE** and **SCALE** are calibration constants in the model must be empirically determined.

As the testing process continues, individual probabilities change depending on whether any errors are discovered. The rate at which the probability changes is dependent on the component complexity and the amount of testing the component has undergone. The starting position of the component on the probability distribution curve determines the initial probability of failure and the rate of change of it's probability because of the shape of the curve. If the component complexity is high, the rate of change of probability is initially low and after a period of time, increases rapidly with additional error-free testing. After an additional amount of testing, if there are no errors, this rate begins to decrease again and becomes

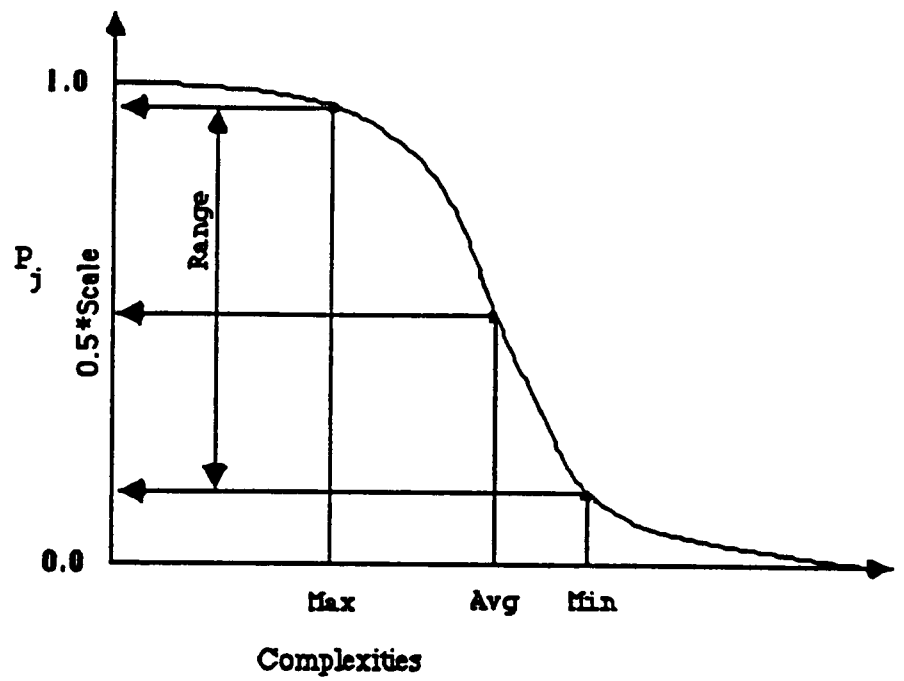


Figure 1 : Assigment of Initial Probabilities

steady at a low value. On the other hand, if the component complexity is low, rate of change of probability is more or less steady and the component would also have a relatively low initial probability of failure with respect to the high complexity components. Figures 2 illustrates the progression on the probability curves for high and low complexity components with testing which does not result in the discovery of an error. Note that the shape of the curve corresponds to the intuition about the change of probability expressed in section 2.1.

2.2.3 Corrections for Error Discovery

A generally accepted observation is that a majority of the errors are discovered in small section of the code. Another indirect observation that follows from this is that among the high complexity components in the system, only a fraction of these components would have a high concentration of errors. Thus, when an error is discovered in a component, its probability of failure should be increased because more errors are likely to be found in this component by additional testing. The increase in probability is determined by the complexity value of the component.

However, if an additional amount of testing does not reveal any errors in a high complexity component, this is a component that, despite having a high complexity, does not have a concentration of errors. For these components one could be sufficiently confident that the component would run reliably and there should be a significant change in it's probability of failure. The positioning of the high complexity components along the probability distribution curve determines the "amount of testing" that is required to come to this conclusion.

Again, if the component complexity is small, then we have probably discovered the only error in the component. However we remain cautious and increase the probability of failure but keeping

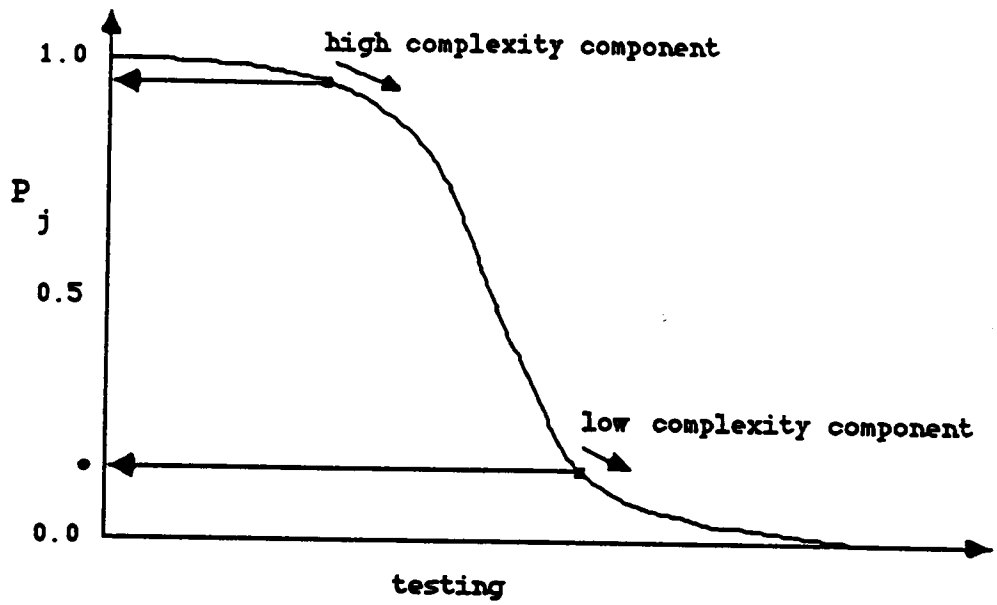
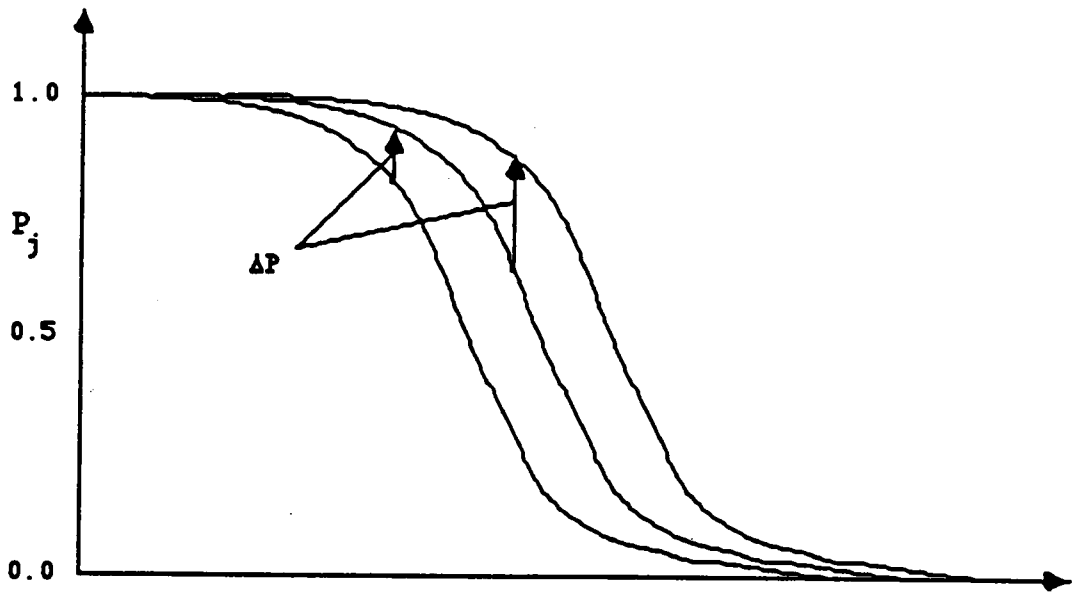


Figure 2 : Progression of Probabilities with Testing

the rate constant. By observing Figure 2, we notice that we can achieve this effect by reassigning the component probability value to an ordinate of the abscissa on the left side of the current abscissa. This amounts the the same things as shifting the probability distribution curve to the right by a few test cases. Figure 3 illustrates this effect.



ΔP : Change in probability with error discovery

Figure 3. : Progression of Component Probability with error correction

2.2.4 System Probability of Failure

From the individual probability of failure of the components, we can derive the system probability of failure. In determining the system probability of failure, at each test case we are implicitly assuming that the behavior of the system we have seen thus far is representative of the future behavior of the system. This is expressed in our calculation of the frequency of execution. We have assumed without stating that at test case i, the degree of testing which a component has undergone is indicative of the amount of testing the component will undergo in the future.

The first step in the derivation is the assignment of failure probabilities to the individual components. At test case i, this is denoted by:

Probability of failure of component j : P_j

The probability of failure of the components determined from the distribution curve in Figure 2 based on the component complexity as discussed in sub-section 2.2.2. Each time the component is executed without failure, the new probability value is determined to be the ordinate **Step** test cases to the right from last test case when it was run. By the same token whenever an error is discovered in the component, the new probability is the ordinate of the test case that is **Back** test cases to the left of the last test case in which the component was run. This procedure captures all the information and analysis that was presented at the component level in the previous subsection (2.2.3).

In order to estimate the number of remaining errors in the component, we need to determine the frequency of execution of the component. A good estimate of this would be to look at the testing

history and count the number of times this component is executed at each point.

Projected Frequency of execution

of component j : $\frac{n_j}{i}$

This is the average number of times component j is executed over i test cases.

The number of remaining errors in the component is determined by estimating the number of times the component is going to be tested in the remaining testing period and then applying the probability of failure estimate.

Projected upper bound on failures

in component j : $(N-i)*\frac{n_j}{i}*p_j$

This is an upper bound on the number of component failures remaining to be discovered. In calculating the upper bound on the number of remaining failures at test case i, we are implicitly assuming that the probability of failure of the component is constant for the remainder of the testing. This is an incorrect assumption. However, since we are determining the upper bound, the analysis is not effected. If subsequently, the component runs without failures, the probability of failure drops and we determine a new upper bound on the component failures.

Once the projected upper bound on the failures is determined for all the components, we can then determine the projected upper bound on the number of remaining failures in the system.

Projected Upper bound on failures

over all components in the system : $\sum_{j=1}^{T}(N-i)*\frac{n_j}{i}*p_j$

This is a summation over all the components in the system and gives an estimate on the total number of failures remaining to be discovered in the system.

To determine the system probability of failure, we have to estimate the total number of component executions i.e. the sum total of the component executions over all components. This is again a projected estimate based on the testing history of each of the components.

Projected number of executions

of component j : $(N-i)*\frac{n_j}{i}$

This is an estimate on the number of component executions over the remaining test cases.

Projected number of executions

over all components : $\sum_{j=1}^{T}(N-i)*\frac{n_j}{i}$

This is a summation over all components and is self-explanatory.

System probability of failure : $\frac{\text{Proj. \# of failures}}{\text{Proj. \# of executions}}$

$$: \quad \frac{\sum_{j=1}^{T} n_j * p_j}{\sum_{j=1}^{T} n_j}$$

System probability of failure is taken as the total number of failures over the total number of executions. This is because even if only one component fails, the entire system is considered to have failed.

Chapter 3

3.0 Simulation Tools and Techniques

3.1 Introduction

The best test of a reliability model's credibility is its accurate and directionally sensitive predictions over a wide range of software systems. Iannino, Musa, Okumoto, and Littlewood [IANN84] have also identified the following criteria for assessment of any reliability model : predictive validity, capability, quality of assumptions, applicability, and simplicity.

Predictive validity is the capability of the model to predict future failure behavior from present and past failure behavior [MUSA82]. There are two ways of viewing predictive validity and are based on the approaches to characterizing the failure process :

1. the number of failures approach and

2. the failure time approach

The first approach, the number of failures experienced over a period of time is characterized by specifying a distribution. We determine the distribution parameters from past data and apply the distribution to predict future failures. In the second approach, we characterize the times between failures. The first approach could be tranformed to the domain of our reliability model (i.e. the number of test cases) and thus will be used in assessing the predictive validity of the reliability model in the subsequent sections.

Capability refers to the ability of the model to estimate with satisfactory accuracy quantities needed by the software managers, engineers, and users in planning and managing software development projects or running operational software systems. Our reliability model employs measurable characteristics like size, complexity,

structure and the testing history to determine reliability. The model thus has the capability for prediction of software reliability in the system design and early development phases. All *assumptions* made in the model development process have been qualified by intution. The model development presented in Chapter 3 attests to its *simplicity*. It is simple enough to be applied to any software system whose complexity metrics can be generated and can be applied both in the development and operational environment thus making it widely applicable unlike the existing 'black box' models which can be applied only after the system has been completely built.

In the absence of actual systems with their complexity metrics, testing history, and failure discovery data, we are forced to resort to simulating the actual systems and employing error seeding and error discovery strategies. In this thesis we present the development of a complete simulator which is used for the credibility analysis of the reliability model presented in Chapter 2.

3.2 Software System Simulator

The purpose of this simulator was to generate hypothetical data for performing the credibility analysis of the reliability model. The software was developed on a UNIX system using the C-language. We also used FORTRAN IMSL mathematics and statistics libraries for generating bivariate gamma random variates.

There are three main functions of the simulator :

1. Generate the System Descriptions based on specified system characteristics.

2. Generate hypothetical error seeding and test case generation information for each system description.

3. Apply the model to each of the system descriptions and their testing history and output the simulation results.

These steps are shown in Figure 4.

3.2.1 Generation of System Description

The system description is represented by complexity-error pairs which as a whole have specified means, variances and correlation. In the absence of any representative data to perform the distribution analysis, we had to choose a distribution function that would demonstrate some representative characteristics of the existing software systems. In our search for such a distribution, we found that the bivariate gamma distribution function has properties that would make it ideal for our purpose. It has been observed that the error distribution among components in a system tends to be highly skewed with a few components having a high concentration of errors and a majority with very few or zero errors [CANNING85]. The gamma distribution function has the advantage that we can control the shape and scale of the distribution independently for each of the variates, thus allowing us the flexibility of modelling different system descriptions uniformly. Also, most of the popular distributions (e.g. Normal, Poisson, Exponential) could be approximated as special cases of the gamma distribution.

Inputs :

1. (mean, variance) or (shape, scale) pairs of the compo nents.
2. (mean, variance) or (shape, scale) pairs of the errors.
3. Number of components in the system (Tcomp).
4. Desired correlation coefficient between complexities and errors.

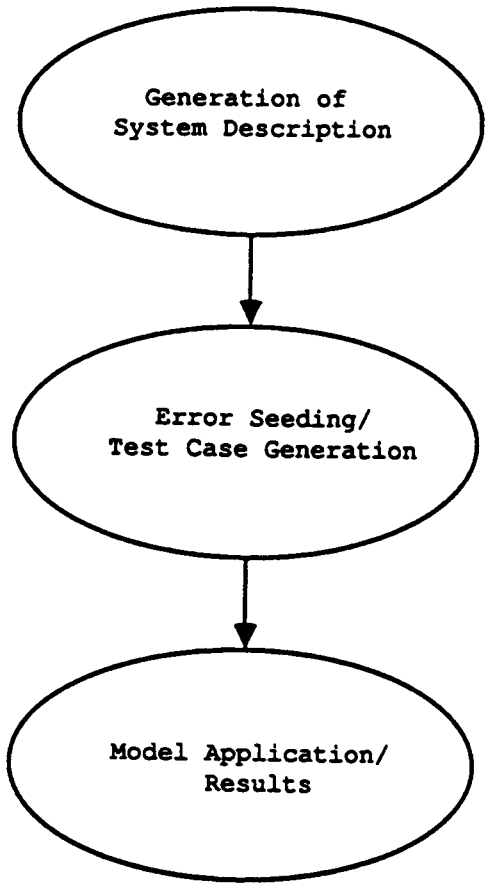


Figure 4 : Software System Simlator

Outputs :

1. A file containing the Tcomp (complexity, errors) pairs.

2. Total number of errors distributed in the system (Terrs).

procedure :

The methods listed in literature for generating bivariate gamma random variates [SCHM80, SCHM82] produce variates in the real domain. However, we require the variates to be in the integer domain. So, the modified algorithm truncates the random variates generated from the algorithm and recalculates the (mean, variance) parameters for the complexities and the errors in the components. If these parameters fall within the 10% domain of their specified values, the system description is accepted.

Figure 5 shows the inputs to and outputs from the bivariate gamma random variate generator. This corresponds to the first phase of the software system simulator as shown in Figure 4. Figure 6 gives the equations for the standard gamma density function and the relationship between (mean,variance) and (scale,shape) parameters. Figure 7 illustrates some representative shapes of the gamma distribution function.

3.2.2 Error Seeding and Testing Distribution

The process that introduces defects into code and the testing selection process that determines which code is being executed at any time are both dependent on a large number of time-varying variables. The tester has partial control over the environment and thus it is possible that the selection of components could be planned or manipulated to some extent during the testing process. However, the introduction of errors into the code and the relationship between

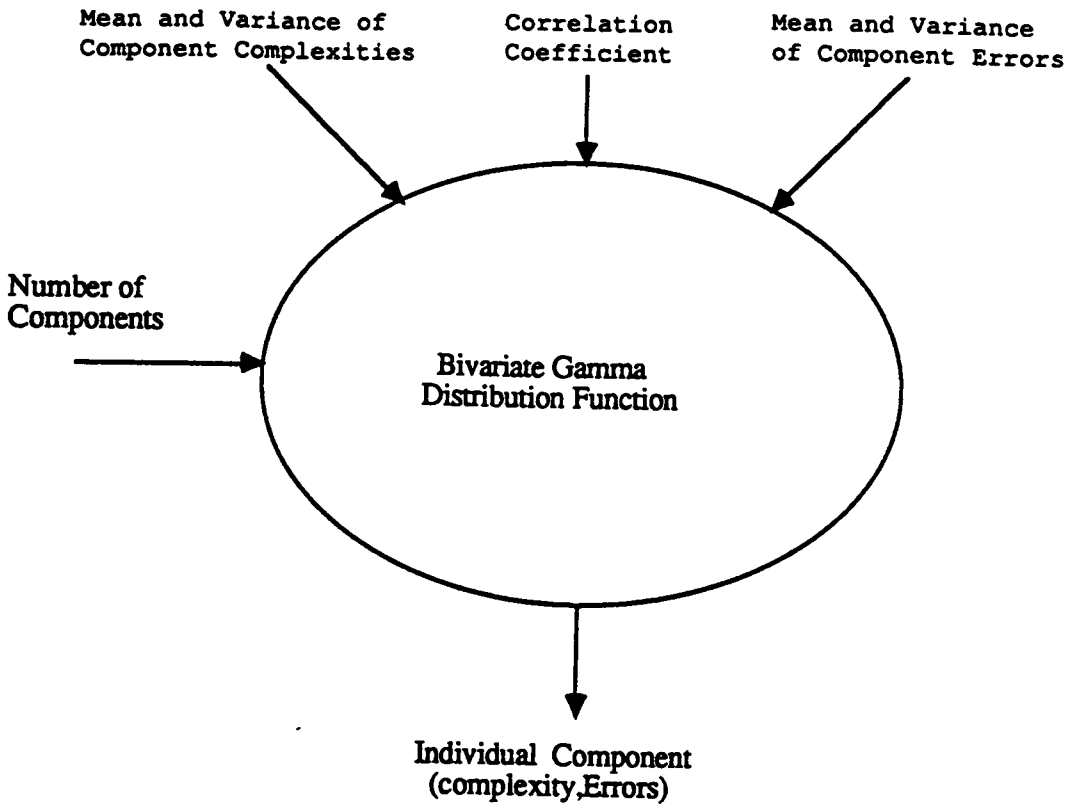


Figure 5 : Generation of System Description

$$f(x) = \begin{cases} \dfrac{\beta^{-\alpha} x^{\alpha-1} e^{-x/\beta}}{\Gamma(\alpha)} & \text{if } X > 0 \\ 0 & \text{otherwise} \end{cases}$$

where

$$\Gamma(\alpha) = \int_0 t^{z-1} e^{-t} dt$$

and

α = Shape Parameter

β = Scale Parameter

Mean $(\mu) = \alpha\beta$

Variance $(\sigma^2) = \alpha\beta^2$

Figure 6 : Gamma Density Function

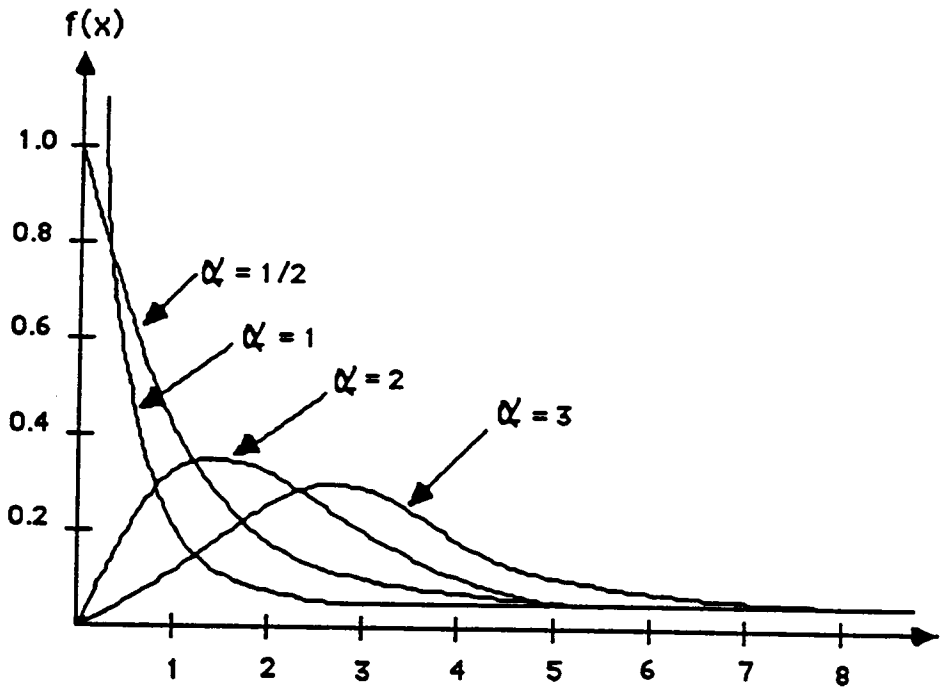


Figure 7 : Gamma Density Functions with Shape Parameter 1.

input states and code executed are usually both sufficiently complex processes to make deterministic prediction of failure impractical [MUSA87]. Consequently, a deterministic selection of input states will not have a perceivable deterministic effect on reliability. Therefore, modelling testing and error discovery as a random process is still a reasonable approach. The testing and error distribution patterns are first generated using a random number and mapping it over the test case range. This constitutes a simulation run, and we then apply the model and determine the predicted system probability of failure and the actual system probability of failure.

3.2.2.1 Testing distribution

As was discussed earlier, even though the tester has partial control over the environment and could influence the selection of components for execution during a test case, there will not be any significant deterministic effect on the prediction of software failure. In addition, since we do not have actual software systems on which we can apply the model, testing approaches such as functional testing are not of consequence here. The number of test cases over which a component is executed and the components that are executed on a particular test case are based on random number generators.

The testing scenario is described by the following :

Inputs :

1. Number of test cases over which the component testing is to be distributed (N).

2. Total number of Components in the system under testing (T).

Outputs :

Matrix of numbers (terr(i,j)) whose rows represent the test cases and columns represent components. Location terr(i,j) is 1 if component j is run in test case i.

Procedure :

For each component j in the system, a uniform random number generator is called to determine the number of test cases in which the component is to executed. This number is then distributed over the test cases by calling a random number generator and linearly mapping the random number over the testing range. At this point, the number of times a component is executed in a given test case is ignored and if there are any overlaps during the mapping, the random number generator is called again. Figure 8 illustrates the sequence of operations.

3.2.2.2 Error Distribution

The system description generator returns complexity-error pairs and the errors in each of the components are to be distributed over the test case range. An obvious fact is that the total number of errors to be distributed should be less than the total number of test cases following our assumption that at most one error is discovered per test case.

The error distribution scenario is described by the following :

Inputs :

1. Number of test cases (N).
2. Number of components (T).
3. Number of errors in each component (e(j)).
4. Testing distribution (terr(i,j)) generated earlier.

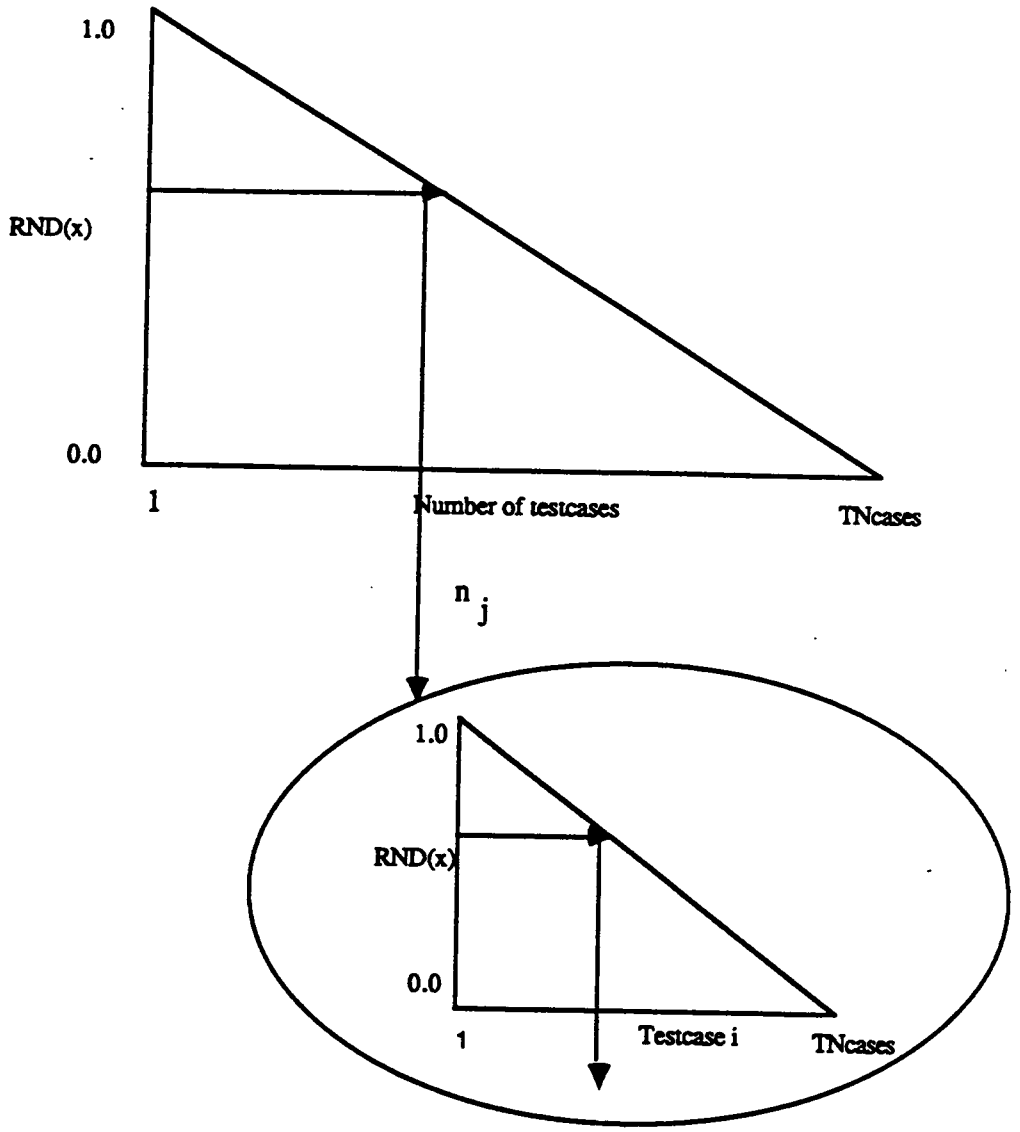


Figure 8 : Generation of test case executions for component j

Outputs :

An array cerr(i) of length N whose elements indicate if an error occurs in test case i, and if so, in which component.

Procedure :

The errors in the system are distributed over the testing sequence using a uniform random number generator. For each compo

nent, j, a random number is generated and mapped over the number of test cases. If the number maps to a test case which has not been marked for error discovery in some other component, the test case array is updated to indicate that component j fails at test case i. The number of errors to be distributed is then decremented. If more than one error maps to the same test case, the mapping is discarded the random number generator is called again. This follows our assumption that at most one error is discovered per test case.

3.2.3 Simulation Output

Once the testing and error distribution tables have been determined, the model is ready to be simulated on a test case by test case basis. The simulator generates the probability of failure data using the model developed in Chapter 3. Initially, we determine the probability of failure of the individual components based on the component complexities. This involves finding a position on the normal cumulative distribution function for each component depending on its complexity. The probabilities of failure of each component are tracked as the simulation proceeds from test case to test case. The probability of failure of components change only with testing. This ensures that components which remain untested in the system will retain their assigned probabilities and, thus, if a large portion of the system remains untested, the system probability of failure remains

high. The system probability of failure is then determined from the individual component probabilities as per the analysis presented earlier. For comparison, we also compute the actual probability of system failure. The actual probability of failure of the system is defined as the ratio of the number of errors remaining to be discovered to the number of test cases remaining to be executed.

3.2.4 Output Statistics

In order to make a comparison between the actual system probability and the predicted system probability and relative performance of the model over different system descriptions, we generated statistics from the output data of the simulation and performed the Z-tests of significance. We have identified three statistics which could be suitable measures of the degree of closeness of the actual and predicted system failure probabilities :

- Absolute difference between the actual and model estimates of probability(|Pact-Pmod|/Pact).
- Count of the number of test cases in which the predicted probability of failure is within 10% of the acutal probability of failure.
- Count of the number of test cases in which the predicted probability of failure is within 25% of the actual probability of failure.

To discount for the possibility of anomalies due to the pseudo-random number generators, the simulation for each system description was duplicated 49 times and the Z-test of significance was applied to make comparisons of the model performance over different system descriptions. The number of duplications chosen is a matter of convenience as the only requirement was that it should be large enough (>>30) so that the Z-tests of significance could be applied.

In the next section we present the hypothesis testing procedure for determining if two population means are different. In the present context, population constitutes a simulation run of the system description. i.e. a population is the test statistics collected over the set of duplications. A complete description of the hypothesis testing can be found in any statistics text and hence only a condensed treatment is offered here.

3.2.5 Z Tests for Differences Between Two Population Means

In performing the hypothesis tests between two populations one may assume that both populations are normal and both samples are normal. This is a realistic assumption since many population distributions are well approximated by a normal curve. As an illustrative example, a normal distribution approximates a binomial distribution when the sample size is very large. Similarly, a poisson distribution also approximates to a normal distribution. Also, we assume that the population variances are known.

However, when sample sizes are large (size>>30), the above assumption need not be true, as, according to the Central Limit Theorem, even if the sample is not normal, the sample mean will have an approximately normal distribution and so will be an arithmetic difference of the means of two non-normal samples. We can also approximate the population variances with the sample variances and arrive at a large-sample test statistic. This is a standard test statistic and hence no justification is provided here.

Effectively, the Z statistic is a normalization of the sample mean differences($\overline{X} - \overline{Y}$). This statistic tests the hypothesis that two population means are the same. Therefore, the complete description for the test procedure would be :

Null Hypothesis $H_0 : \mu_1 - \mu_2 = 0$

Test Statistic $$Z = \frac{\overline{X} - \overline{Y} - (\mu_1 - \mu_2)}{\sqrt{\frac{S_1^2}{n} + \frac{S_2^2}{n}}}$$

Where

$\overline{X}, \overline{Y}$	:	Sample Means
S_1^2, S_2^2	:	Sample Variances
μ_1, μ_2	:	Population Means
n	:	Sample Size

Alternative Hypothesis	Rejection Region for Level α Test
$H_a : \mu_1 - \mu_2 > 0$	$Z \geq z_\alpha$
$H_a : \mu_1 - \mu_2 < 0$	$Z \leq -z_\alpha$

α is the probability of Type I error (probability of rejecting the null hypothesis when it is true). Typically, α is chosen as 0.1 and the boundary value for the Z statistic at this level of confidence is 1.29.

Chapter 4

4.0 Credibility Analysis

4.1 Introduction

In order perform the credibility analysis of the model, 20 different system descriptions were generated. As mentioned earlier, a system description is completely defined by six parameters : mean and variance of errors, mean and variance of complexities, total number of components(Tcomp) and the correlation coefficient between errors and complexities. The following are the abbreviations used in the Tables :

Tcomp	:	Total number of components in the system.
Terrs	:	Total number of errors in the system
MeanC	:	Mean of the complexities in the system
VarC	:	Variance of the complexities in the system
MeanE	:	Mean of the errors in the system
VarE	:	Variance of the errors in the system
Correl	:	Correlation coefficient between errors and component complexities

The system descriptions are organized into five different sets. Within each set, four of the variables are fixed (to within 10%) and the fifth is varied. The following tables list the five sets. Each table lists four system descriptions with four of the parameters fixed and the fifth varying. The numbering scheme for the tables used in this chapter is as follows

Each table is numbered X.Y :

Where X is the set number[1..5] and Y gives associated tables for the set. The possible values for Y are :

Y = 1 : System descriptions used for the simulations.

Y = 2 : Output Statistics for the simulations.

Y = 3 : Hypothesis testing results.

In addition to the terminology defined above, the following abbreviations represent the output measures from the simulation analysis :

Relative Differences : Relative differences between the actual and model probabilities. This is the sum total of the absolute relative difference between the actual and the modal estimates over all the test cases

10% Envelope : Count of the number of test cases in which the predicted probability falls within 10% of the actual probability.

25% Envelope : Count of the number of test cases in which the predicted probability falls within 25% of the actual probability.

Zr : Computed test statistic for relative differences

Z10 : Computed test statistic for 10% envelope

Z25 : Computed test statistic for 25% envelope

A discussion on the output measures was presented in Chapter 4. The following sections present an analysis of the output data on a set by set basis.

4.2 Model Behavior with Changing Variance of Complexities :

Looking at Table 1.2 we notice that as the variance of the complexities increases, the relative difference between the actual and the predicted probabilities is consistently decreasing, and the count of the number of points within the 10% and 25% envelopes is also consistently increasing. Computation of the test statistics in Table 1.3 also confirm this observation in quantitative terms. Even though the tests of significance between systems 3 and 4 for the relative differences and the 25% envelope do not show a significant difference, the 10% envelope does indicate that the model probability curve for system 4 is much closer to the actual curve than system 3. This implies that the model is performing well with increasing variance of the component complexities. Increasing variance of complexities implies a greater discrimination of the components with respect to their complexities. This is consistent with our model hypothesis that the component complexities are related to errors in the system. A number of metrics associated with complexity have a high variance and thus could be used effectively in the model.

We notice that the variance of the output measures seems to be unusually high. This implies that some simulations do well and some simulations do very poorly. The variance of the 10% measure seems to increase consistently while the variance of the 25% measure seems to drop consistently. This anomaly between the 10% and the 25% measures can be attributed to the scattering effect in the mid-complexity region. Figure 9 is a plot of the distribution of the complexity-errors pair from typical system descriptions. The plot indicates a certain relationship between the complexities and errors. However in the mid-complexity regions, we see a very high perturbation or scattering. The 25% measure has the containing effect and thus is more insensitive to this scattering.

TABLE 1.1 : System Descriptions with Varying Variance of Complexities

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl
1	250	179	88.72	1931.71	0.716	1.323	0.6348
2	250	173	91.44	2498.31	0.692	1.261	0.6056
3	250	178	88.04	3693.40	0.712	1.253	0.6242
4	250	173	88.71	7336.25	0.692	1.293	0.6313

Table 1.2 : Output Statistics from Varying Variance of Complexities

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	25.13	24.96	31.14	324.20	126.12	1957.16
2	22.22	40.56	42.63	782.76	157.28	3002.61
3	18.09	18.66	67.36	911.00	194.51	1230.98
4	17.39	33.95	95.22	1316.82	197.63	943.70

Table 1.3 : Computed Z statistic from Varying Variance of Complexities

System descriptions	Zr	Z10	Z25
1-2	-2.519	2.417	3.097
2-3	-3.756	4.207	4.004
3-4	-0.677	4.131	0.468

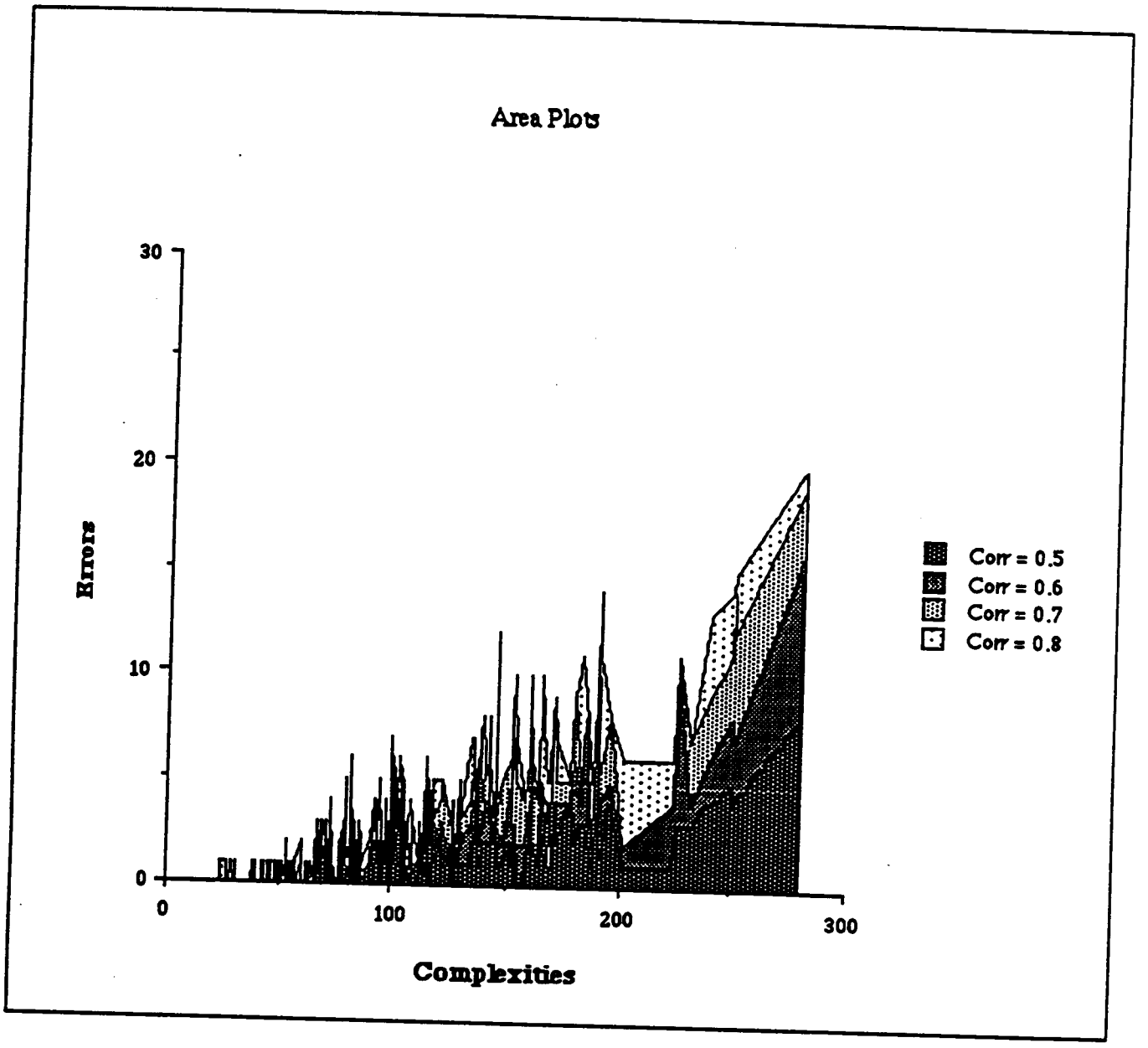


Figure 9 : Area Plots for Systems with different Correlation Coeffients

4.3 Model Behavior with Changing Variance of Errors

As we change the variance of the errors across the system descriptions (Table 2.1), we notice that even though in absolute terms the model seems to improve in performance with decreasing variance, performing the tests of significance shows that the differences are not significant. There is no apparent consistent pattern in the statistical measures. This behavior could be attributed to the fact that the change in variance of the errors over which the model is tested is not very significant in absolute terms. In actual systems, it has been observed that the variance of the errors tends to be quite high with respect to the mean [NASA82a]. The orders of magnitude difference between the means of the complexities and errors coupled with a relatively large variance of errors with respect the mean could not be modelled by any of the existing standard bivariate discrete or continous distributions(bivariate normal, poisson, beta and gamma distributions). All the distributions that we have investigated, implicitly assume that both the variates are of the same order of magnitude. For this reason, we are not able to make any quantitative judgements about the model behavior with respect to the variance of errors.

TABLE 2.1 : System Descriptions with Varying Variance of Errors

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl
1	250	173	91.44	2498.31	0.692	1.261	0.6056
2	250	179	90.92	2711.07	0.716	1.635	0.6091
3	250	186	91.74	2750.90	0.744	2.158	0.6111
4	250	189	88.72	2619.44	0.756	2.344	0.5865

Table 2.2 : Output Statistics from Varying Variance of Errors

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	22.22	40.56	42.63	782.76	157.28	3002.61
2	21.51	29.11	46.87	1011.94	154.42	2703.38
3	23.77	23.12	33.04	444.44	127.73	1639.74
4	23.97	21.76	28.34	416.43	129.14	1905.83

Table 2.3 : Computed Z statistic from Varying Variance of Errors

System descriptions	Zr	Z10	Z25
1 - 2	0.590	-0.701	-0.110
2 - 3	-2.185	2.569	2.830
3 - 4	-0.211	1.122	0.278

4.4 Model Behavior with Changing Correlation Coefficient

Table 3.3 shows the results from performing simulations over system descriptions with varying correlation coefficients (Table 3.1). As the correlation coefficient increases, the relative difference between the actual and model probabilities is decreasing and the count of the number of points within the 10% and 25% envelopes in also consistently increasing. Table 3.3 gives the test statistics for all the three measures. The 10% measure does quite well and passes all the test requirements. However, the 25% measure statistically does not behave consistently. The relative difference measure is not satisfactory.

We had, however, expected the model to be quite sensitive with respect to the correlation coefficient. One explanation for the insensitivity of the model lies in the orders of magnitude difference in numbers between the complexity values and the errors in the system. Also, a look at the complexity-error pairs for system descriptions with various correlation coefficients (Figure 9) shows that there is a significant overlap in the point spread between different correlations, especially for complexity values in the midrange. Point spread is the deviation of the errors from the regression line of the system description. This makes it harder for the model to discriminate effectively over the range of correlations.

One significant problem we have discovered is that the correlation coefficient alone is insufficient to accurately define the relationship between the errors and complexities. Figures 10 and 11 show two scatter diagrams which have the same correlation coefficient. However, at a certain complexity, though the expected number of errors is the same, the deviation from this expected value could be significant. Figure 11 shows a significantly larger deviation than Figure 10. Most of the system descriptions that we generated have scatter plots similar to Figure 11 and thus the output results tend to be inconclusive.

TABLE 3.1 : System Descriptions with Varying Correlation Coefficient

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl
1	250	186	88.60	2634.83	0.744	1.342	0.5026
2	250	173	91.44	2498.31	0.692	1.261	0.6056
3	250	171	90.44	2816.88	0.684	1.288	0.7135
4	250	170	90.03	2856.96	0.680	1.194	0.8125

Table 3.2 : Output Statistics from Varying Correlation Coefficient

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	24.62	31.03	32.65	427.89	133.67	2423.19
2	22.22	40.56	42.63	782.76	157.28	3002.61
3	21.02	29.03	53.46	1259.18	161.65	2733.45
4	20.80	45.29	65.26	1155.33	177.22	1839.84

Table 3.3 : Computed Z statistic from Varying Correlation

System descriptions	Zr	Z10	Z25
1 - 2	-1.990	2.007	2.241
2 - 3	-1.006	1.678	0.411
3 - 4	-0.180	1.680	1.611

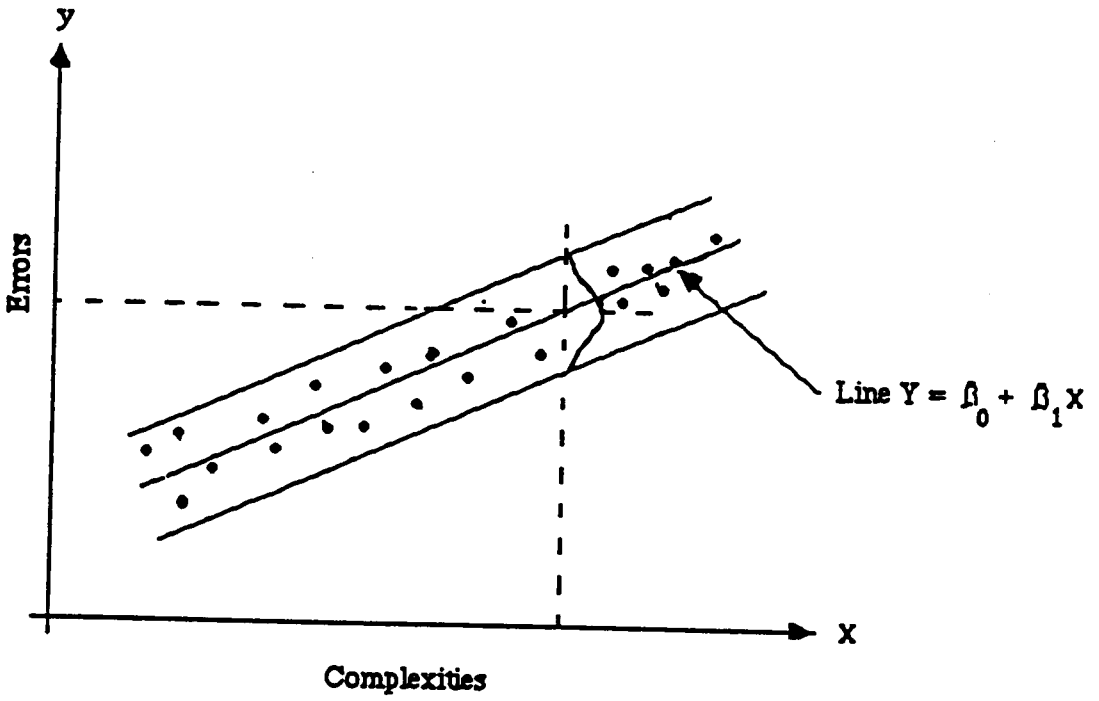


Figure 10 : Regression line with Small Variance

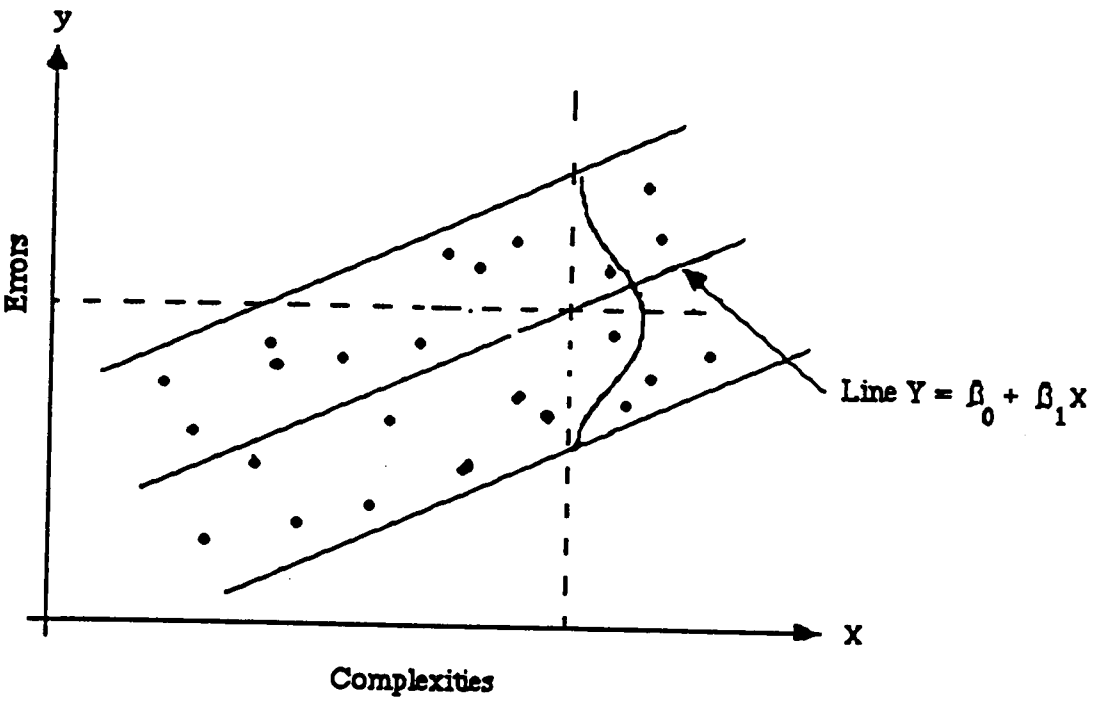


Figure 11 : Regression with a Large Variance

4.5 Model Behavior with Changing Mean of Complexities

Table 4.3 gives the simulation results from varying the average component complexities. The table does not show any consistent behavior for the relative differences and the 10% Envelope. The tests of significance for these statistical measures in Table 4.3 also do not show any significant difference. However, the 25% envelope does seem to show that the accuracy of model predictions drops with increasing mean complexities. One reason could be the scattering effect mentioned earlier. This effect would be more pronounced with higher mean complexities.

TABLE 4.1 : System Descriptions with Varying Mean of Complexities

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl
1	250	190	76.72	2866.87	0.760	1.238	0.5776
2	250	173	91.44	2498.31	0.692	1.261	0.6056
3	250	182	98.92	2696.06	0.728	1.310	0.6424
4	250	175	114.32	2651.74	0.700	1.258	0.5905

Table 4.2 : Output Statistics from Varying Mean of Complexities

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	22.55	35.47	45.06	795.28	149.38	2966.80
2	22.22	40.56	42.63	782.76	157.28	3002.61
3	23.05	19.49	35.63	851.70	141.00	2090.69
4	24.53	17.51	31.85	458.93	122.97	1525.24

Table 4.3 : Computed Z statistic from Varying Mean of Complexities

System descriptions	Zr	Z10	Z25
1 - 2	-0.111	-0.428	0.750
2 - 3	0.704	-1.212	-1.570
3 - 4	0.243	-0.731	-2.090

4.6 Model Behavior with Changing Mean of Errors

Tables 5.2 and 5.3 give the simulations results from varying the mean errors in the system. The results show a significant change in the model performance with changes in the mean error content in the system. All three measures indicate a change in the model performance with mean error content. A closer look at the simulation results shows that the model performs best for system description 1. However, the model was tuned using this system description as the base. Therefore, the rapid deterioration of the model with other system descriptions suggests that the model might have to be re-tuned for the system description with a different mean error content.

The model was retested against the system descriptions after rescaling the initial probability assignments. This could be easily achieved in the model by changing the SCALE parameter. Table 5.4 lists the range of calibration constants for which the model performs consistently well in all three output measures. We should notice that there is a consistent trend in the scale factor though, at higher mean of errors, the best scale parameter tends to be in a large range of values. Table 5.5 gives the simulation results after making a selection of the SCALE parameter from Table 5.4. The selection criteria is more or less the midpoint of each of the ranges. The results show that the model performs quite well for all the system descriptions after calibrating the SCALE parameter. Therefore, it can be concluded that, if the mean error content in the system can be estimated by other techniques or by historical experience, the model can be calibrated accordingly to give accurate predictions.

Comparing Tables 5.2 and 5.5, we notice that the Relative Difference measure between the actual and model prediction seems to be consistenly dropping. The 10% and the 25% envelopes also seem to be capturing a significant portion of the curve. The tests of significance conducted on Table 5.5 also confirm that changes are significant.

TABLE 5.1 : System Descriptions with Varying Mean of Errors

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl
1	250	173	91.44	2498.31	0.692	1.261	0.6056
2	250	255	91.79	2874.85	1.020	2.116	0.5853
3	250	306	90.98	2813.82	1.224	2.142	0.6035
4	250	388	95.40	2870.32	1.552	2.135	0.5797

Table 5.2 : Output Statistics from Varying Mean of Errors

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	22.22	40.56	42.63	782.76	157.28	3002.61
2	35.20	26.15	21.34	408.67	52.28	595.99
3	43.58	25.70	6.59	140.52	27.44	622.28
4	54.17	22.62	1.14	27.55	8.69	271.23

Table 5.3 : Computed Z statistic from Varying Mean of Errors

System descriptions	Zr	Z10	Z25
1-2	11.10	-4.322	-12.27
2-3	8.16	-4.411	-4.98
4-5	10.66	-2.940	-4.98

TABLE 5.4 : System Descriptions with Varying Mean of Errors and the SCALE factor

Sys#	Tcomp	Terrs	MeanC	VarC	MeanE	VarE	Correl	SCALE
1	250	173	91.44	2498.31	0.692	1.261	0.6056	1.1-1.2
2	250	255	91.79	2874.85	1.020	2.116	0.5853	0.6-0.7
3	250	306	90.98	2813.82	1.224	2.142	0.6035	0.3-0.5
4	250	388	95.40	2870.32	1.552	2.135	0.5797	0.1-0.4

Table 5.5 : Output Statistics from Varying Mean of Errors for a Chosen Scale Factor

Sys#	Relative Differences		10% Envelope		25% Envelope	
	Mean	Var	Mean	Var	Mean	Var
1	18.21	73.06	121.36	959.90	200.71	722.32
2	19.80	69.60	133.49	797.96	189.18	841.41
3	22.22	62.10	141.81	536.35	178.94	316.87
4	25.49	64.36	128.33	726.30	170.00	221.06

Table 5.6 : Computed Z statistic from Varying Mean of Errors for a Scale Factor

System descriptions	Zr	Z10	Z25
1 - 2	0.93	2.025	-1.940
2 - 3	1.470	1.595	-2.069
4 - 5	2.035	-2.650	-2.416

Table 6.0 : Summary of Results

Factor	Directionally Correct			Significance		
	Zr	Z10	Z25	Zr	Z10	Z25
Correlation Coefficient	3	3	3	1	3	2
Mean of Errors	3	3	3	3	3	3
Mean of Complexities	2	3	2	0	0	2
Variance of Errors	2	2	2	1	1	1
Variance of Complexities	3	3	3	1	3	2

Chapter 5

5.0 Conclusions

5.1 Summary and Comments on the Model Behavior

Summarizing the results we have obtained from simulations, the model seems to be directionally correct to changes in the variance of the complexities and mean of the errors. The Z-test also shows that the changes are significant. The model also does well with respect to the correlation coefficient albeit below our expectations. The model is directionally correct with varying mean of complexities. However, tests of significance have not been very good. We cannot make judgements on the model performance with respect to the variance of errors for the reasons stated in Chapter 5. Table 6.0 summarizes the results of the model performance with respect to the various system parameters. Table 6.1 is a qualitative assessment of the model behavior with respect to each of the system parameters in terms of directionality, tests of significance and model sensitivity with changes to the system parameters

5.2 Future Work

One of the constraints in our modelling was our inability to accurately generate bivariate gamma random variables in the integer domain. We have used a modified version of the bivariate gamma random number generator in the real domain. Consequently, we notice a scattering effect in the complexity-error distribution at mid-range complexities. This significantly effects the sensitivity of the model. If a better algorithm can be found and implemented, more accurate comments can be made about the model behavior with respect to the mean of the complexities and variance of errors.

Secondly, in our analysis of the model behavior, we found out that the correlation between errors and complexities alone was in

sufficient to completely describe a system. There may be other measures which can more accurately capture the relationship between errors and complexities and should be investigated. These measures could then be used in place of correlation in the model.

Thirdly, the scattering effect can also be investigated further if we can find a methodology for controlling this effect.

Finally, other metrics such as the execution time complexity metrics (actual execution time statistics of the individual components) and calendar time metrics when validated could also be incorporated into the model. A comparison of the reliability model performance with respect to the already existing 'black box' reliability models could then be performed.

In conclusion, the complexity based modelling of software systems to determine reliability seems to be a credible approach.

Table 6.1 : Model Performance

Factor	Directionally Correct	Significance	Sensitivity
Correlation Coefficient	Yes	Good	Good
Mean of Errors	Yes	Good	Good
Mean of Complexities	Yes	Fair	Fair
Variance of Errors	-	-	-
Variance of Complexities	Yes	Good	Good

References

[AVIZ77] Avizienis A. and L.Chen, "On the implementation of n-version programming for software fault-tolerance during execution", Proc IEEE COMPSAC, 1977, pp. 149-155.

[CANN85] Canning James Thomas , "The Application of Structure and Code Metrics Large Scale Systems", Phd thesis, VA TECH, May1985.

[GOEL178] Goel, A. L. and K. Okumoto, "Bayesian Software Prediction Models : An Imperfect Debugging Model for Reliability and Other Quantitative Measures for Software Systems", RADC-TR-78-155, July 1978.

[IANN84] IEEE Transactions on Software Engineering, "Criteria for Software Reliability Model Comparisons", SE-10(6), pp. 687-691.

[KAFU81a] Kafura D., and Henry, S., "Software Quality Metrics based on Interconnectivity", The Journal of Systems and Software, Vol. 2,1981, pp. 121-131.

[LEW88] Ken S. Lew, Tharam S. Dillon, Kevin E. Forward, "Software Complexity and Its Impact on Software Reliability" IEEE Trans. on Software Engineering, Vol. 14, No. 11, November 1988, pp. 1645-1655.

[LITTLE74]Littlewood, B., and J. L. Verrall, "A Bayesian Reliability Model with a Stochastically Monotone Failure Rate", IEEE Transcations on Reliability, R-23(2), 1974, pp. 108-114.

[MCCA76] McCabe, T. J., "A Complexity Measure", IEEE Transactions on Software Engineering, SE-2, December 1976, pp 308-320.

[MCCL78] McClure, C., "A Model for Program Complexity Analysis" Proceedings 3rd International Conference on Software Engineering, Atlanta, GA., 1978

[MUSA79d]Musa, J. D., "Validity of Execution Time Theory of Software Reliability" IEEE Transactions on Reliability, R-28(2), 1979, pp. 181-191.

[MUSA87] John D. Musa, Anthony Iannino, Kazuhira Okumoto, "Software Reliability: Measurement, Prediction, Application"McGraw-Hill Series in Software Engineering and Technology,1987.

[MORAND75]Moranda P. B., "Predictions of Software Reliability during Debugging" Proceedings Annual Reliability and Maintainability Symposium, Washinton DC,1975, pp. 327-332.

[NASA82] "Guide to Data Collection", Software Engineering Laboratory Series SEL-81-101, August 1982.

[RAND75] B. Randell, "System Structure for software fault-tolerance", IEEE Trans. on Software Eng., Vol. SE-1,1975, pp. 220-232.

[SCHIK73] Schick, G. J. and R. W. Wolverton, "Assesment of Software Reliability", Proceedings Operations Research, 1973, pp. 395-422.

[SCHIK78] "An Analysis of Competing Software Reliability Models", IEEE Transactions on Software Engineering, SE-4(2), 1978, pp.104-120.

[SCHM82] Schmeiser, B. W. and R. Lal, "Bivariate Gamma Random Vectors"Operations Research vol 30, No. 2, March-April, 1982.

[SCHM80] Schmeiser, B. W. and R. Lal, J. ,"Squeeze Methods for Generating Gamma Variates", Am. Statist. Ass., 75:,1980, pp. 679-682.

[SHOOM72]Shooman, M. L., "Probabilistic Models for Software Reliability Prediction", Statistical Computer Performance Evaluation, Academic, New York, 1972, pp. 485-502.

[SHOOM76]Shooman M. L., "Structural Models for Software Reliability Prediction" , Second International Conference on Software Engineering, IEEE, pp 268-280, Oct 1976.

[SUKERT77]Sukert A. N., "A Software Reliability Modeling Study", Rome Air Development Center, Technical Report RADC-TR-76-247, Rome, NY, 1976.

[WAGON73]Wagoner, W. L., "The Final Report on a Software Reliability Measurement Study", The Aerospace Corp., Report No. Tor-0074-(4112)-1, August 15, 1973.

The vita has been removed from the scanned document