ABSTRACT

Title of Dissertation : ANALYSIS OF ERRORS IN SOFTWARE RELIABILITY PREDICTION SYSTEMS AND APPLICATION OF MODEL UNCERTAINTY THEORY TO PROVIDE BETTER PREDICTIONS

Susmita Ghose, Ph.D 2006

Dissertation Directed By: Dr. Carol S. Smidts,
Associate Professor,
Department of Mechanical Engineering.

Models are the medium by which we reflect and express our understanding of some aspect of reality, a particular unknown of interest. As it is virtually impossible to grasp any situation in its entire complexity, models are representations of reality that are always partial resulting in a state of uncertainty or error. However the question of model error from a pragmatic point of view is not one of accounting for the difference between models and reality at a fundamental level, as such difference always exists. Rather the question is whether the prediction or performance of the model is correct at some practically acceptable level, within the model's domain of application.

Here lays the importance of assessing the impact of uncertainties about predictions of a model, modeling the error and trying to reduce the uncertainties associated as much as possible to provide better estimations.

While the methods for assessing the impact of errors on the performance of a model and error modeling are well established in various scientific and engineering disciplines, to the best of our knowledge no substantial work has been done in the field of Software Reliability Modeling despite the fact that the inadequacy of the present state and techniques of software reliability estimation has been recognized by industry and government agencies. In summary, even though hundreds of software reliability models have been developed, the software reliability discipline is still struggling to establish a software reliability prediction framework.

This work intends to improve the performance of software reliability models through error modeling. It analyzes the errors associated with a set of five software Reliability Prediction Systems (RePSs) and attempts to improve their prediction accuracy using a model uncertainty framework. In the process, this work also statistically validates the performances of the RePSs. It also provides a time and cost effective alternative to performing experiments that are required to assess the error form which is integral to the process of application of the model uncertainty framework.

ANALYSIS OF ERRORS IN SOFTWARE RELIBILITY PREDICTION SYSTEMS AND APPLICATION OF MODEL UNCERTAINTY THEORY TO PROVIDE BETTER PREDICTIONS

By

Susmita Ghose

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:
Associate Professor Carol Smidts, Chair
Professor Ali Mosleh
Professor Mohammad Modarres
Assistant Professor Michel Cukier
Professor Marvin Zelkowitz

DEDICATION

To my parents, Prativa Rani Ghose and Sudhir Kumar Ghose

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Dr. Carol Smidts for her immense help in guiding my research. As an advisor she assisted me in every aspect from research brainstorming to writing this dissertation.

I am fortunate to have been able to work on this project with a talented and dedicated team of UMD researchers consisting of Dr. Ming Li, Dr. Bin Li, Dr Avik Sinha, Dr Dongfeng Zhu, Yuan Wei, Anand Ladda, Wende Kong andYing Shi. I would like to thank them for their help and support to this project.

I would like to thank Dr. Mohammad Modarres, Dr. Ali Mosleh, Dr. Marvin Zelkowitz, Dr. Michel Cukier for agreeing to be on my committee. I am grateful to Dr Mosleh, Dr Zelkowitz and Dr Cukier for their inputs.

Finally I would like to thank Alok Priyadarshi, my husband, for helping me intellectually and emotionally throughout the research.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 The Issue

Despite the fact that hundreds of software reliability models have been developed to date [Smidts02], the software reliability discipline is still struggling to establish a software reliability estimation and prediction framework.

This work intends to improve the performance of software reliability models through error modeling. It analyzes the errors associated with a set of five software Reliability Prediction Systems (RePSs) [Smidts00, Smidts02] and attempts to improve their prediction accuracy using a model uncertainty framework.

Models are the medium through which we reflect and express our understanding of some aspect of reality, a particular unknown of interest. As it is virtually impossible to grasp any situation in its entire complexity, models are representations of reality that are always partial. In other words, what we know about the true nature of the unknown of interest is generally incomplete, resulting in a state of uncertainty. This uncertainty is termed as "error" and is defined as the difference between the true value of the unknown of interest and the value predicted by the model. The error in model predictions can arise from uncertainties in the values assumed by the model parameters, uncertainties and errors associated with the structure of the model stemming from simplifications, assumption and approximations.

Model uncertainty arises in natural sciences and engineering when in addressing a situation of interest when there is [Droguett02]

- No plausible model
- A single model, generally accepted, but not completely validated
- Conceptually accepted and validated models, but of uncertain quality of implementation
- A single model covering some but not all relevant aspects of the problem
- Multiple plausible models, none of which is completely validated
- Competing theories with contradictory predictions
- Multiple models each covering different aspects of the reality of interest
- Composite models formed from sub-models with different degrees of accuracy and credibility

In evaluating the uncertainties associated with a model various sources of information might be available. This includes comparison of actual measurements and results of experiments directly or indirectly related to model predictions. Information concerning a particular model itself may also be available.

Thus, the central question is what we can say about the unknown of interest given all the available sources of information. Can the information be used to obtain a better performance of the model?

Model performance is to be seen in the context of its objective and scope. The question of model error/uncertainty from a pragmatic point of view is not one of accounting for the difference between models and reality at a fundamental level, as such difference always exists. Rather the question is whether the prediction or

performance of the model is correct at some practically acceptable level, within the model's domain of application.

Here lays the importance of assessing the impact of the uncertainties about predictions of a model, modeling the errors, and trying to reduce the uncertainties associated as much as possible to provide better estimation.

While the methods for modeling the errors to assess the impact of errors/uncertainties on the performance of a model are well established in various scientific and engineering disciplines [Albert01, Badar05, Cromlry02, Cho04, Bierman95, Bessler03], to the best of our knowledge no work has been done in the field of Software Reliability Modeling.

The inadequacy of the present state and techniques of software reliability estimation has been recognized by industry and government agencies. For instance, the nuclear industry usually uses IEEE STD 7-4.3.2-1993, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations." While the Nuclear Regulatory Commission (NRC) endorsed this standard in Regulatory Guide 1.152, Revision 1 (January, 1996), it did not endorse Section 5.15, "Reliability" as a sole means of meeting the Commission's regulations for reliability of digital equipment used in safety systems. The applicable Section 5.15 of the standard states "when qualitative or quantitative reliability goals are required, the proof of meeting the goals shall include software used with hardware." The NRC did not endorse that section because there is no general agreement that a measurement methodology currently exists that provides a credible method to measure software reliability [NRC96, NRC97, Smidts00].

The aircraft industry standard for software is RTCA/DO-178B [RTCA92], "Software Considerations in Airborne Systems and Equipment Certification." It states in section 12.3.4 'Software Reliability Methods' that "currently available methods do not provide results in which confidence can be placed to the level required for this purpose." Hence this document does not provide guidance for software failure rates.

In summary, despite the fact that hundreds of software reliability models have been developed to date [Smidts02], the software reliability discipline is still struggling to establish a software reliability estimation and prediction framework. This work intends to improve the performance of software reliability models through error modeling. It analyzes the errors associated with a set of five software Reliability Prediction Systems (RePSs) [Smidts00, Smidts02] and attempts to improve their prediction accuracy using a model uncertainty framework.

1.2 The Objectives

The objectives of this work are as follows

- Statistically validate the performance of a set of five software Reliability Prediction Systems (RePSs). A detailed discussion on the RePSs is provided in Chapter 3.
- Analyze and determine the nature of the errors associated with the RePSs, classify the errors and once that is accomplished, apply the results to the model uncertainty framework to better the performances of the models. This should allow a better estimation of reliability.

- Provide a platform for generalizing model uncertainty problems to the software reliability modeling domain and suggest an approach to improve model performance using the model uncertainty approach.
- Suggest alternate methods which are cost and time effective to determine the nature of errors. Determining the nature of the errors is integral to applying the model uncertainty framework and is sometimes infeasible from a cost or time perspective. The objective is also to validate these alternate methods.

1.3 The Approach

Simulations were carried out to determine the nature of the errors. Traditionally the nature of the errors is determined experimentally. That includes the extensive and difficult task of designing the experiment in a way that counter threats to validity, and executing it. Each experiment requires a minimum number of data points (the larger the number, the better) in order to statistically validate it. This approach is expensive not only from a cost perspective but also from a time perspective. Sometimes carrying out an experiment is just not feasible due to lack of resources. Simulation is an alternative which provides a solution to the above problems. Moreover, simulation allows us to use a wide range of inputs. This not only provides a broader spectrum of possibilities but also acts as a catalyst for sensitivity analysis of the inputs/values.

This work also validates the results obtained from the simulation on the nature of the error models and their prediction accuracies. This validation was important to establish the level of accuracy of each RePSs and confirm/reject/refine the error

models associated with the RePSs. This validation was also important to confirm whether the assumptions made in the simulation process can be validated and whether the simulation process can be generalized as an alternative to an experimental approach. The validation was also significant from the perspective of generalizing the assumptions and the simulation approach to construct error models for simulation from other software reliability models. Therefore an in-vitro experiment was designed [Field03], [Hughes71] involving eight different applications to validate the findings from simulation.

1.4 The Contents

The rest of this work is organized as follows. Chapter 2 provides a detailed discussion on model and model uncertainty. The discussion includes the definitions of a model, the reasons for model uncertainty, the taxonomy of model uncertainty, the model uncertainty framework and the applications of the framework. It also provides examples of domains where error modeling has been applied.

Chapter 3 provides a detailed discussion on the theory of RePSs and the process of construction of the five RePSs from five software engineering measures: Defect Density, Bugs per Line of Code, Requirements Traceability, Function Point and Test Coverage. This provides the foundation for the construction of error models for the simulation.

Error models for each of the RePSs are then built for the simulation. Chapter 4 enumerates the importance of simulation and provides the rationale and assumptions

behind the construction of error models for simulation. It then summarizes the results and discusses them.

Chapter 5 presents an experiment involving eight software applications to establish the level of accuracy of each RePSs and confirm/reject/refine the error models associated with the RePSs. This includes the experiment design, the objectives, the hypotheses, the threats to validity and the execution process of the experiment. It also provides the experimental results and statistically analyzes them. The results obtained from simulation are compared to the experimental results and are analyzed for similarity.

Chapter 6 applies the model uncertainty framework to the results obtained from the experiment and the simulation. It then presents a new and more robust software reliability prediction. The predictive ability of the new model is also assessed.

Finally Chapter 7 summarizes and concludes this research. It identifies the contributions of this work, provides the limitations, and then suggests possible future avenues of research.

Chapter 2 Literature Review

This chapter presents an overview of recently published research on model uncertainty and the applications of error modeling to different scientific and engineering disciplines. It also then presents an overview of the research done to assess the impact of errors in software reliability modeling domain.

2.1 Model and Model Uncertainty

This section presents a discussion on model and model uncertainty. The discussion includes definitions of a model, the reasons for model uncertainty, the taxonomy of model uncertainty, the model uncertainty framework and the applications of the framework.

2.1.1 Definition of "model"

The Concise Oxford Dictionary [Thompson95] defines model as one of the following:

- a representation in three dimensions of an existing person or thing or of a proposed structure, especially on a smaller scale (a model train)
- a simplified (often mathematical) description of a system, etc., to assist calculations and predictions
- a particular design or style of a structure or commodity.

Why is modeling an integral part of our lives? That is because modeling is necessary to explain any real event. Modeling takes place in the quest of handling a specific problem.

2.1.2 Uncertainty

Uncertainty is a reflection of the lack of knowledge. There have been different schools of classification of uncertainty. According to one classification, and the most widely accepted, uncertainty is of basically two types:

- Epistemic uncertainty (also known as subjective uncertainty, knowledge uncertainty, reducible uncertainty): it arises from lack of knowledge about the state of reality under study and is thus a property of the analysts performing the study [Helton96A]
- Aleatory uncertainty (also known as stochastic uncertainty, variability uncertainty, irreducible uncertainty): it arises because the reality under study can behave in many different unpredictable ways and is thus a property of reality [Helton96A]. That is, there are aspects of reality that are inherently stochastic. This is due to the inherent variability of nature. This also leads to the assertion that such a type of uncertainty is irreducible even in principle, i.e., further knowledge or better understanding of the natural phenomena underlying the process under investigation is not useful.

2.1.3 Model Uncertainty

As discussed before, reducing reality into a model inevitably results in an error, reflecting the discrepancies between the reality portion of interest and its model representation. These errors can be associated with the structure of the model stemming from simplifications, assumption and approximations or due to uncertainties in the values assumed by the model parameters or due to errors in the

measurement process itself. This error can be viewed as a measure of how good a model is in representing reality.

2.1.4 Quantifying Model Uncertainty

A number of approaches for quantifying model uncertainty have been proposed in the literature with varying degrees of complexity, strength of theoretical foundation, and capability in addressing different model uncertainty situations. Among them, two approaches have seen wider practical applications. One approach involves averaging of predictions of multiple models, the Model Averaging method and the other uses model adjustment factors and is also known as Correction Factor method or the Uncertainty Factor Approach. The following sections provide a brief theoretical description of the Model Averaging and Uncertainty Factor approaches and a discussion regarding each methodology's major features and highlight their respective advantages and disadvantages.

2.1.4.1 Model Averaging Method

The Model Averaging (MA), also known as Alternate Hypotheses approach [Zio96] or the $P\{M_i\}$ approach [Mosleh95], considers each available model a representation of a set of plausible hypotheses about a system that, in light of available evidence, provides predictions about a common quantity of interest. The predictions of available and plausible models are then combined probabilistically via mixture of distributions, thus requiring a set of mutually exclusive and collectively exhaustive models.

The MA approach is employed either implicitly or explicitly in various domains. de Finetti [deFinetti72], Davis [Davis79], Draper et al. [Draper87], Draper [Draper95, Draper98], Hodges [Hodges87], Madigan and Raftery [Madigan94, Madigan96], Laskey [Laskey95, Laskey96], Chatfield [Chatfield96] have used it in the fields of statistics and decision theory. Apostolakis [Apostolakis90, Apostolakis95] addressed the issue of model uncertainty through this approach in probabilistic risk assessments. Chhibber et al. [Chhiber91] applied the MA procedure to the quantification of model uncertainty in the context of distributed environmental contamination. Zio and Apostolakis [Zio96] used the MA framework in the performance assessment of radioactive waste depositories, particularly dealing with models of groundwater flow and contaminant transport. The MA procedure has also been suggested and applied in other areas. Hoeting et al. [Hoeting98] provided an extensive discussion of the methodology and applied it in dealing with the uncertainty in the treatment of primary biliary cirrhosis of the liver and from the prediction of percent body fat using 13 alternate body measurements in a multiple regression model.

2.1.4.1.1 The MA Theory

The *Model Averaging* (MA) approach, as mentioned, combines the predictions of various plausible models probabilistically via a mixture of distributions and computes the average value [Zio96].

A mathematical model may be represented [Droguett02] as $x_i = M_i(\Theta_i, S_i)$, where x_i is the prediction of the model about a reality aspect of interest, S_i

represents the model's form reflecting a set of assumptions and simplifications encoded into the mathematical model M_i, and $\Theta_i = (\theta_1, \theta_2, \ldots)$ is a finite set of model parameters. In a general case of a discrete set of n models Ω, each model $M_i(\Theta_i, S_i)$, i=1,2…n, represents an alternate form of S_i with given set of parameters Θ_i. Each model in the set Ω provides an estimate about the quantity of interest X in the form of a predictive probability distribution $p(x \mid M_i) = p(x \mid \Theta_i, S_i)$.

The Model Averaging (MA) method treats the model $M_i(\Theta_i, S_i)$, as a variable or a parameter and integrates over uncertainty about both the model form S_i and model parameters Θ_i [Draper95]. Given available evidence E and the set Δ of possible models' forms, the posterior distribution of quantity X is provided by the standard Bayesian estimator

$$p(x \mid E) = \sum_{\Delta_i} [\int_{\Theta_i} p(x \mid E, \Theta_i, S_i) p(\Theta_i, S_i \mid E) d\Theta_i] \qquad (2\text{-}1)$$

representing a weighted average of the conditional predictive distributions $p(x \mid E, M_i)$ using the posterior model probabilities $p(M_i \mid E)$ as weights. Now, writing the posterior model probabilities $p(M_i \mid E)$ as

$$p(\Theta_i, S_i \mid E) = p(\Theta_i \mid E, S_i) p(S_i \mid E) , \qquad (2\text{-}2)$$

the posterior distribution $p(x \mid E)$ can be rewritten as follows:

$$p(x \mid E) = \sum_{\Delta_i} [\int_{\Theta_i} p(x \mid E, \Theta_i, S_i) p(\Theta_i \mid E, S_i) p(S_i \mid E) d\Theta_i] \qquad (2\text{-}3)$$

The posterior probability distribution for a model M_i can then be obtained by Bayes theorem. The probability of a specific model M is given in terms of its form S_i, i.e., $p(S_i \mid E)$.

The MA framework is an intuitive approach. Furthermore, all probability distributions involved in the MA approach can be updated via Bayes' theorem. In particular, given new evidence, posterior model probability distributions can be obtained.

2.1.4.1.2 The Pros and Cons

However, the MA approach has several drawbacks. First of all, the concept of assessing a probability distribution over the available model set necessarily leads to the question of how to interpret the probabilities $p(S_i)$. This interpretation is implied by, the two fundamental assumptions on which the MA approach is based:

- the set of alternate models should be mutually exclusive and
- the set of alternate models should be collectively exhaustive. Simply put, the model probabilities should sum up to one (as required by the summation in eq. (2-3)).

Both of these assumptions are hardly satisfied in practice. The collective exhaustiveness, for example, implies that not only the probability attributed to a model, say $p(S_i)$, is "correct" but also that the correct model be one of the alternate models $M_1, \ldots, M_n$. As stated by Winkler [Winkler96], "in most real world situations, the possible existence of a correct model is questionable at best." Moreover, the set of plausible models is unavoidably incomplete.

The mutual exclusiveness assumption of the plausible models $M_1, \ldots, M_n$ implies that it is not possible to explicitly model dependence among the set of

alternate models under the MA framework. This assumption imposes a strong restriction in practical situations.

An illustration of cases where models are not mutually exclusive and collectively exhaustive has been provided by Bier [Bier95].

2.1.4.2 Uncertainty-Factor Methodology

The Uncertainty-Factor (UF) method [Siu85], also known as Error-Factor approach [Chhiber91], accounts for model uncertainty by modifying the prediction given by a single "best" model (also called the reference model) by means of a correction factor, which is usually uncertain, in order to estimate the true value of a quantity of interest [Siu92]. In terms of applications, Siu et al. [Siu92] applied the UF method in the context of fire risk assessment, more precisely in the estimation of the velocity of flame spread over a horizontal cable tray. Zio and Apostolakis [Zio96] present an application of the uncertainty-factor methodology to the performance assessment of radioactive waste depositories in which it is used as a second-stage model uncertainty assessment procedure following the identification of the single "best" available model via quantification of posterior model probabilities under the MA approach.

2.1.4.2.1 The UF Theory

The Uncertainty-Factor (UF) approach [Siu92] consists of introducing an adjustment directly on the predictions provided by a single model. Formalizing, let $x_i = M_i(\Theta_i, S_i)$, where x_i is the prediction of the model about a reality aspect of interest, S_i represents the model's form reflecting a set of assumptions and

simplifications encoded into the mathematical model M_i, and $\Theta_i = (\theta_1, \theta_2, \ldots)$ is a finite set of model parameters. A factor ξ is introduced which may be multiplicative (ξ_m) or additive (ξ_a) so that the assessment about the unknown quantity of interest X *is* given by

$$X = X_M + \xi_a \tag{2-4}$$

in the additive case, or

$$X = X_M / \xi_m \tag{2-5}$$

for the multiplicative case.

The correction factor (ξ_a or ξ_m) modifies the prediction of the model M in order to arrive at the true value of the quantity of interest X.

Considering the simple case where X is a deterministic single valued quantity and adopting the multiplicative uncertainty-factor model given by eq. (2-4), in order to determine the uncertainty distribution of ξ_m, Siu [Siu92] suggests that if the model's parameters Θ_i are known, then the ratio of X_M/X (or equivalently $X_M - X$ in the additive case) is fixed for a given experiment (where each experiment corresponds to any situation in which data can be gathered). However, ξ_m is likely to vary from experiment to experiment. This variability is expressed by the correction factor ξ_m population variability distribution (its uncertainty distribution). Now if $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$ is a finite set of m parameters, as per Bayes' Theorem,

$$\pi(\Lambda \mid E) = \frac{L(E \mid \Lambda)\pi_0(\Lambda)}{\int_{\Lambda} L(E \mid \Lambda)\pi_0(\Lambda d\Lambda} \tag{2-6}$$

where E is the evidence on model performance, E.

where $L(E|\Lambda)$ is the likelihood function and $\pi_o(\Lambda)$ is the prior distribution on the set of parameters $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$. Under the restriction of n independent experiments, the likelihood function is given by

$$L(E \mid \Lambda) = \prod_{i=1}^{n} f(\xi_{m_i} \mid \Lambda) \qquad (2\text{-}7)$$

where $\xi_{mi} = \frac{X_{M_i}}{X_i}$ or $\xi_{ai} = X_{M_i} - X_i$ in the additive case, and X_i is the measured value of the quantity of interest X in experiment i and X_{Mi} is the corresponding predicted value by model . The posterior population variability, $f(\xi_m|E)$, is obtained by averaging over all possible values of the parameter set Λ ,

$$f(\xi_m \mid E) = \int_{\Lambda} f(\xi_m \mid \Lambda)\pi(\Lambda \mid E)d\Lambda \qquad (2\text{-}8)$$

The modeler's final goal is to estimate the probability distribution of the quantity of interest X given the prediction provided by the model and any additional performance data E. This distribution can be obtained from eq. (2-5)

2.1.4.2.2 The Pros and Cons

A positive aspect of the uncertainty-factor approach with a population variability distribution for the error term ξ_m is that it allows for the use of a model outside its intended domain of application. The uncertainty-factor tries to correct the predictions provided by model M by means of the correction factor ξ , and make them applicable to the situation at hand which the model was not initially designed to handle. Thus, ξ can be interpreted as a factor that indicates how far are the current application's conditions and assumptions to those for which the model was intended

for [Zio96]. The uncertainty-factor method allows for the incorporation of experimental data into the assessment of model uncertainty.

In this work, since one of the objectives is to validate each of the five models, it is desired to use each of the models separately. Also, it can not be assumed that the models are mutually exclusive and collectively exhaustive (Section 2.1.4.1.2). Therefore, a generalized version of the Uncertainty Factor Approach suggested by [Droguett02] is used in this work to analyze the five RePS models. This approach is Bayesian in nature, simple to use, allows us to incorporate experimental data and update the estimates made by the models. It can be applied to additive and multiplicative error models. Chapter 6 discusses the approach in greater details.

2.2 *Error Modeling*

In this section, the importance of error modeling is reiterated and the applications of error modeling to various scientific and engineering domains are discussed. Error is the difference between the reality and the model representation of the reality behavior.

A mathematical model may be represented as $x_i = M_i(\Theta_i, S_i)$, where x_i is the prediction of the model about a reality aspect of interest, S_i represents the model's form reflecting a set of assumptions and simplifications encoded into the mathematical model M_i, and $\Theta_i = (\theta_1, \theta_2, \ldots)$ is a finite set of model parameters. Now *error* is defined as the difference between the real value of unknown quantity of interest X and x_i, the value predicted by M_i. Errors are analyzed to determine their

nature and their sources. This information is then used to model the errors. Modeling the errors provides a better understanding of the impact of the errors and how they affect the real value X in different scenarios. By different scenarios it means different contexts in which the models are used, their impact if there are changes in the parameters etc. It provides a sensitivity analysis of the models.

Error modeling is widely conducted in the field of mechanical and industrial engineering. Various approaches are proposed for machine tool error modeling and compensation. Some of these approaches are described below.

Different mathematical models such as coordinate transformation are given by [Schultschik77], [Ferreira86]. Other approaches, including empirical, trigonometric, and error matrix methods are proposed by [Ferreira86]. A simulation study is conducted in [Wang06] to illustrate an error compensation procedure.

[Badar05] presents an adaptive sampling procedure, which uses manufacturing surface error patterns and optimization search methods to reduce sample size, while improving accuracy. Surface errors for different processes are quantified and validated using previously published models. The initial points for sampling are identified through such a characterization of the process and its effect on the workpiece. These sampled points are fit using the least-squares method to complete the form evaluation. Points are added to the initial set using optimization search heuristics. The final tolerance value obtained is compared with that obtained from a large population sample to check the accuracy. With such an adaptive approach, it is proposed that the number of points sampled is potentially less than that which would be expected to achieve the same level of accuracy using traditional sampling

methods. This paper demonstrates the error modeling and validation aspects of this adaptive sampling procedure.

Error modeling has been widely used in the field of construction engineering. [Cho04] proposes a correction to improve the position error in automated construction manipulators. Hydraulically actuated construction equipment is rapidly being retrofitted with robotic control capabilities by several major manufacturers. However, position control errors caused by several factors are significant in these types of construction equipment. Errors are amplified if the manipulator and its operator must measure and locate objects in the equipment's fixed reference frame. Both mechanistic and statistical approaches to correcting position errors are possible. [Cho04] reports a statistical approach validated through experiments with a computer-controlled large-scale manipulator (LSM). The LSM is sufficiently representative of several types of construction equipment to be able to serve as a general test bed. In the regression analysis, three factors which are measurable in real time: distance, hydraulic pressure, and payload, are varied to determine their influence on position errors in the LSM. It is shown that with an integrated multivariable regression model, about 30% of the mean positioning error of the LSM can be reduced without the use of fixed external reference systems. The model is implemented as simple, real-time regression equations.

Error modeling has also been used extensively in the fields of biometrics and medical sciences. Two models of disease progression among healthy persons with a history of a precancerous lesion and the errors associated with them are studied by. [Goldie03]. Evaluating cancer screening often requires modeling the underlying

disease process and not the observed disease, particularly in the absence of direct evidence linking screening to a survival benefit. Two models with four basic health states (disease free, presence of a precancerous lesion, presence of cancer, dead), are studied and the errors associated are analyzed. This modeling error's magnitude is examined under a variety of assumptions and finally certain errors when modeling the underlying disease process in evaluating screening programs for cancers associated with precancerous states are removed.

The assessment and management of exploited fish and invertebrate populations is subject to several types of uncertainty. This uncertainty translates into risk to the population in the development and implementation of fishery management advice. Here, risk is defined as the probability that exploitation rates will exceed a threshold level where long term sustainability of the stock is threatened. [Fogarty96] studies the different sources of errors: (a) stochasticity in demographic rates and processes, particularly in survival rates during the early life stages; (b) measurement error resulting from sampling variation in the determination of population parameters or in model estimation; and (c) the lack of complete information on population and ecosystem dynamics. Short term stochastic projections are then made accounting for uncertainty in population size and for random variability in the number of young surviving to enter the fishery.

Error modeling is also widely used in the fields of economics and trade and commerce. [Bessler03] examines dynamic relationships among wheat prices from five countries for the years 1981-1999. Error correction models and directed acyclic graphs are employed with observational data to sort-out the dynamic causal

relationships among prices from major wheat producing regions: Canada, the European Union, Argentina, Australia, and the United States. The empirical results show that Canada and the U.S. are leaders in the pricing of wheat in these markets and that the U.S. has a significant effect on three markets excluding Canada.

Error modeling has been applied in the field of safety critical systems like nuclear power plants, in the aviation industry etc. Air traffic control automation synthesizes aircraft trajectories for the generation of advisories. Trajectory computation employs models of aircraft performances and weather conditions. In contrast, actual trajectories are flown in real aircraft under actual conditions. Since synthetic trajectories are used in landing scheduling and conflict probing, it is very important to understand the differences between computed trajectories and actual trajectories. [Jackson96] examines the effects of aircraft modeling errors on the accuracy of trajectory predictions in air traffic control automation. Three-dimensional point-mass aircraft equations of motion are assumed to be able to generate actual aircraft flight paths. Modeling errors are described as uncertain parameters or uncertain input functions. A typical trajectory is defined by a series of flight segments with different control objectives for each flight segment and conditions that define segment transitions. A constrained linearization approach is used to analyze trajectory differences caused by various modeling errors by developing a linear time varying system that describes the trajectory errors, with expressions to transfer the trajectory errors across moving segment transitions. A numerical example is presented for a complete commercial aircraft descent trajectory consisting of several flight segments.

[Brannigan93] studies the computerized fire risk assessment models and proposed a set of guidelines for their regulatory use. To do that he analyzes the errors associated with the models. He outlined the sources of errors in these models: mainly the assumptions taken into account while building these models like equivalence of buildings, special cases like arson etc.

Even though error modeling application traverses a variety of fields, to the best of our knowledge it has not been used in the field of software reliability modeling. However the impact of errors in software reliability models has been studied. [Brocklehurst90] analyzes the predictive accuracy of several software reliability growth models using "u-plot", which allows a user to estimate the relationship between the estimated reliability and the true reliability using the past performance data. Then the future estimates are improved by a process of recalibration. Recalibration is done assuming that there is a consistent bias (i.e. consistent over-estimation or under-estimation) between the estimated reliability and the true reliability.

[Li93] argues that there is no way to tell exactly how close an estimation will be to the actual value using the recalibration process mentioned in [Brocklehurst90]. The authors simply measure the bias of the estimation of the software reliability at each past point of time and then take the average of these bias values and deduct this average bias from the model projection. They also show that this simple approach is superior to the recalibration method.

[Matsumoto88] evaluate software reliability growth models in the context of a software project conducted in a university. A compiler was implemented and tested

by five students. The test data was used to evaluate three different software reliability growth models. Evaluation of the three models is based on the magnitude of the relative error of the prediction. The relative error is defined as the ratio of the difference between the actual number of defects detected during the testing process and the number of defects predicted by the models, to the actual number of defects. [Malaiya92] evaluate five different software reliability growth models using 18 datasets collected from a wide variety of software systems. The size of the projects range from 1000 Lines of Code to 1 million Lines of Code and come from different domains. These datasets are used to estimate the parameters of the five models. Evaluation of the models is based on the mean relative error of the prediction. Here also the relative error of a model is defined as the error in the prediction of total number of faults in a specific dataset by a model over the actual number of faults in that dataset.

[Lyu96] discusses a study where linear combination of results, even in their simplest format, appears to provide more accurate predictions. The following strategy is adopted in forming linear combination models.

- Identify a basic set of models (component models): If possible select models whose assumptions are close to the actual environment. Also, select those models whose predictive biases tend to cancel each other. (Models can have optimistic or pessimistic biases)
- Apply certain criteria to ascribe weights to the component models and form a combination model for the final predictions.

The authors experimented with three different models. The authors, in their investigation, found that these models perform well. Moreover, with the data sets that the models were analyzed with, one of the models tended to be optimistic, one pessimistic and one went either way. They also experimented with statically weighted combinations and dynamically weighted combinations. In statically weighted combinations, each component model has a constant weighing which remains the same throughout the modeling process. In dynamically weighted combinations, the weights are dynamically assigned incorporating the latest information on the models.

Even though there has been some study on the evaluation of software reliability models, error modeling has not been done to analyze and then improve the estimations. This work generalizes the domain of application of error modeling approaches to the software reliability field. It analyzes and evaluates the prediction accuracy of Reliability Prediction Systems (RePSs) that are constructed from software measurements like Requirements Traceability, Defect Density, Function Point count, Test Coverage and Bugs per Line of Code. The errors for each of the RePSs are modeled and the impact of the errors for different parameters is determined. The nature of the errors associated with the RePSs (multiplicative or additive) is also determined, which is then applied to the model uncertainty framework to update the estimates.

2.3 *Summary*

This chapter described the error modeling applications and approaches in different scientific and engineering fields. The various approaches to error modeling include empirical, trigonometric, statistical, error matrix methods, simulation studies, coordinate transformation methods etc. Many of the approaches of error modeling were applied in real-time to reduce the errors. The approaches looked at sources of variations/errors and tried to rectify the sources itself. The approaches also looked at different types of errors such as model errors and parametric errors and found that most of the errors resulted from assumptions made by the models and the parameters.

This chapter also discussed various studies on the evaluation of software reliability models. However, it was also noted that error modeling approaches has not been applied to the software reliability modeling field. This work aims to apply error modeling approaches to the software reliability field.

Chapter 3 Literature Review: Software Reliability Modeling and Reliability Prediction Systems (RePSs)

This chapter first enumerates some of the widely used software reliability models and then discusses the advantages of RePSs and the theory behind the construction of the RePSs.

3.1 Software Reliability Models

Before introducing the concept of software reliability, few other concepts need to be understood. The concepts in question are those of errors, faults, and failures. Following are the IEEE [IEEE90] definitions of these concepts.

Errors are human actions that result in the software containing a fault. Examples of such faults are the omission or misinterpretation of the user's requirements, a coding error, etc.

Faults are manifestations of an error in the software. If encountered, a fault may cause a failure of the software. In this work, the term "defect" and "fault" are used interchangeably.

Failure is the inability of the software to perform its mission or function within specified limits. Failures are observed during testing and operation.

Now, *Software Reliability* is defined as the probability that the software will not cause the failure of a product for a specified time under specified conditions; this probability is a function of the inputs to and usage of the product, as well as a function of the existence of faults in the software. The inputs to the product determine whether an existing fault is encountered or not.

Software reliability models may be categorized into Early Prediction Models that can predict the reliability of software during the requirement, design or coding phases of the Software Development Life Cycle (SDLC) and Late Prediction Models that can predict the reliability when comprehensive testing starts [Smidts02].

Models for early prediction are few in number and most models can be categorized in the late prediction category. The Late Prediction Models mostly consists of the Software Reliability Growth Models. Input Domain Models and Error-Seeding Models are also late prediction models. Some of the widely used models are discussed below.

The *Rome Air Development Center (RADC) Reliability Metric* was one of the first early prediction models [ASFC87] to be used. A large range of software programs and related failure data were analyzed in order to identify the characteristics that would influence software reliability. The model identifies three characteristics: the application type *(A),* the development environment *(D),* and the software characteristics (S). A new software is examined with reference to these different characteristics. Each characteristic is quantified, and reliability *R* in terms of number of faults per executable line of code is obtained by multiplying these different metrics

$R = A \times D \times S$ where

The *application type (A)* is a basic characteristic of software. Examples of application types that were initially used by RADC are airborne systems, process control systems, developmental systems (such as software development tools), etc.

An initial value for the reliability of the software to be developed is based only on the application type. This initial value is then modified when other factors characterizing the software development process and the product become available.

Development environment (D) is divided into three categories [Boehm81].

- *Organic mode:* Small software teams develop software in a highly familiar, in-house environment. Most software personnel are extremely experienced and knowledgeable about the impact of this software development on the company's objectives.
- *Semidetached mode:* Team members have an intermediate level of expertise with related systems. The team is a mixture of experienced and inexperienced people. Members of the team have experience with some specific aspects of the project.
- *Embedded mode:* The software needs to operate under tight constraints. In other words, the software will function in a strongly coupled system involving software, hardware, regulations, and procedures.

The *software characteristics (S)* metric includes all characteristics of the software that are likely to impact software reliability like the size, complexity etc.

Software reliability growth models (SRGM) relates the cumulative number of failures experienced during software testing (or the time-interval between software failures) to the test duration.

Some of the widely used SRGMs are discussed below.

Jelinski and Moranda's Model [Jelinksi72]: Jelinksi and Moranda developed one of the earliest reliability models. The main assumptions are:

- All faults in a program are equally likely to cause a failure during test
- The hazard rate is proportional to the number of faults remaining and is piecewise constant i.e it changes at each fault correction by a constant amount but remains constant between corrections.
- No new defects are introduced into the software as testing and debugging occur i.e. debugging is perfect.

Originally, the model assumed only one fault was removed after each failure, but an extension of the model, credited to *Lipow* [Lipow74], permits more than one fault to be removed.

Goel and Okumoto [Goel78] developed a modification of the Jelsinki-Moranda model for the case of imperfect debugging.

Musa Basic Execution Time Model [Musa75] assumes that failures occur as a non-homogeneous Poisson process (NHPP). The important assumption in this model is that the per-fault hazard rate is constant. The per-fault hazard rate is defined as the ratio of initial failure intensity to the number of faults inherent in the code [Musa87]. Failure intensity function is defined as the instantaneous rate of change of the expected number of failures with respect to time. Moreover, in this model, Musa postulated that software reliability theory should be based on execution time, which is the actual processor time utilized in executing the program, rather than on calendar time. Hence, failure intensity is measured in terms of numbers of failures per unit (CPU) time. A Bayesian approach to software reliability measurement was taken by *Littlewood and Verrall* [Littlewood73]. Most models postulate that the hazard rate is a function of the number of faults remaining,

whereas as Littlewood and Verrall modeled it as a random variable. One of the parameters of the distribution of this random variable is assumed to vary with the number of failures experienced which characterizes reliability change. The authors proposed various functional forms for the description of this variation. The values of the parameters of each functional form that produce the best fit for that form is determined and then the functional forms are compared and the best fitting form is selected.

In *Musa-Okumoto Logarithmic Poisson Execution Time Model* [Musa84], the underlying software failure process is modeled as a logarithmic Poisson process wherein the total number of failures in the system is "infinite in infinite time". The intensity function decreases exponentially with the number of failures. This model assumes that repair of the first failure has the greatest impact in reducing failure intensity and the impact of each subsequent repair decreases exponentially.

The Delayed S-shaped SRGM was originally proposed by Yamada et al. [Yamada83] and is a simple modification of the NHPP to obtain an S-shaped growth curve for the cumulative number of failures detected. This model's software fault detection process can be viewed as a learning process in which the software testers become familiar with the testing environments and tools and as time progresses, these testers' skills gradually improve and then level off as the residual faults become more difficult to uncover.

Input-domain models consider the software input space from which test cases are chosen and the studied quantity is the probability that an input datum randomly chosen according to the operational profile, will lead to a failure. By

recording the output results for a series of test cases, this probability can be estimated using some statistical sampling techniques. Two well known input domain models are Nelson's model [Nelson78] and Ramamoorthy and Bastani's model [Ramamoorthy82].

Fault seeding models assume that a known number of faults, called "seeded" faults, are inserted into the software and both seeded faults and inherent faults are detected during testing. The number of faults remaining after testing can then be estimated from the numbers of seeded faults and inherent faults uncovered during the testing. Mills fault seeding model [Mills72] is an example of this kind of models.

3.2 *Theory of RePSs*

This section discusses the pros and cons of early and late prediction models and the importance of RePSs and the theory behind it. Early prediction models are of paramount importance since they provide early identification of cost overruns, resource allocation, software development process issues, trade-off and risk analysis, optimal development strategies, etc. Unfortunately, research in this area has been sparse and results are not universally accepted due to a lack of systematic validation and the rapid obsolescence of results due to shifts in software engineering paradigms [Li06]. Late prediction models also have inherent flaws. The main issue is the need for exorbitant amounts of testing (especially in case of safety critical systems) and the availability of failure data. As an example, to assure 10 E-15 probability of failure per demand one will need to run and order of 10E14 test cases

[Butler93]. This, at the rate of 0.1 seconds per test case, would require more than 10E5 years of continuous, uninterrupted testing [Butler93].

The inadequacy of the present state and techniques of software reliability estimation has been recognized by industry and government agencies [NRC96, RTCA92]. In summary, despite the fact that hundreds of software reliability models have been developed to date [Smidts02], the software reliability discipline is still struggling to establish software reliability estimation and prediction model as the hardware reliability discipline did years ago. This is mainly due to the fact that most of these models either require failure data information and trends observed in the failure data or need exorbitant amount of testing and assume that the failure data is available (especially in case of safety critical systems) to predict reliability. Moreover most of these models have not been verified and validated extensively.

Reliability prediction systems, [Li00, Smidts00] RePSs provides an alternative to these models. RePS are constructed from software measurements. The measurements are obtained from different phases of the software development life cycle (SDLC) and hence do not rely upon the availability of just the failure data. In addition, these predictions do not require extensive amount of testing which in turn saves time and money. From an organization's perspective, the time required to estimate these measures and their cost effectiveness make them ideal candidates for reliability estimation. The RePSs can be used alone or with existing methods/techniques of reliability estimation as a check of conformance.

Fig 3-1 depicts the constitution of a RePS [Li00, Smidts00]. Construction of a RePS starts with the "Measure", which is also the "root" of a RePS. "Support

measures are identified to connect the measure to reliability. The set of the "measure" and "support measures" constitutes a RePS. The "model" between "Reliability" and RePS is also termed "software reliability model".

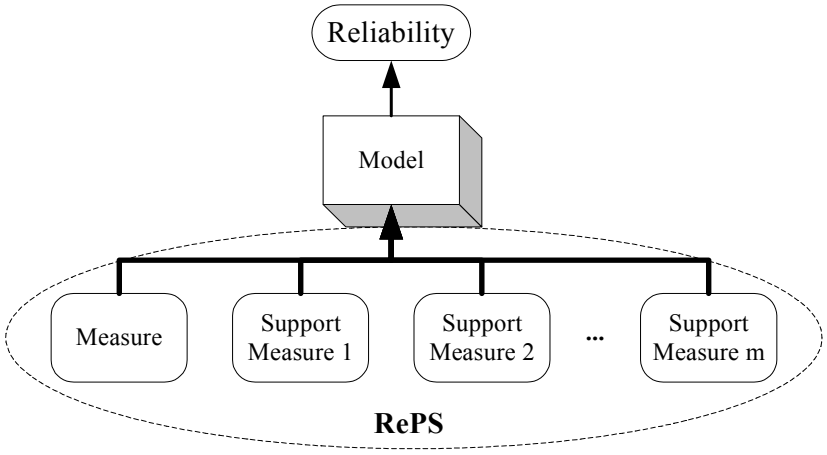


Figure 3-1 RePS Constitution [Li00]

The set of five RePS taken into account in this study are Defect Density, Bugs per Line of Code, Function Point, Requirements Traceability and Test Coverage. [Li00, Li04, Smidts00, Smidts04] discusses the RePS construction from these measures in details. However a summary of the RePSs is provided below in order to set the foundation for the simulation.

3.2.1 Defect Density RePS

Defect density is defined as the number of defects remaining unresolved in an application divided by the number of lines of code in the application. The Defect Density RePS is constructed taking into account the defects discovered by independent inspection. However, please note that this is an approximation of the defect density measure. Although the defect density ratio is traditionally meaningful as an indicator of the quality of development, only the defects themselves that are

detected through inspection is taken into account for the reliability estimation [Smidts00].

Software fails due to the defects introduced during the development process. A defect leads to a failure if it meets the following conditions: first, it needs to be triggered (executed); then such execution should modify the computational state; and finally such abnormal state should propagate to the output and manifest itself as an abnormal output, in other words, a failure [Thompson93, Voas92].

The PIE concept [Voas92] was used to describe such failure mechanism in [Smidts00]. The acronym PIE corresponds to the above three program characteristics: the probability that a particular section of a program (termed "location") is executed (termed "execution" and noted as E), the probability that the execution of such section affects the data state (termed "infection" and noted I) and the probability that such an infection of the data state has an effect on program output (termed "propagation" and noted P). Thus the failure probability per demand p_f is given in (3-1).

$$p_f = \oint_i P(i) * I(i) * E(i) \qquad (3\text{-}1)$$

where

P(i) the propagation probability for the *i*th defect

I(i) the infection probability for the *i*th defect

E(i) the execution probability for the *i*th defect.

A simple, convenient and effective method using an extended finite state machine model (EFSM) [Wang93] can be used to determine failure probability.

EFSMs describe a system's dynamic behavior using hierarchically arranged states and transitions. A state describes a condition of the system; and the transition visually describes the system's new state as a result of a triggering event.

3.2.2 Test Coverage RePS

In this section the RePS construction from Test Coverage is examined. Test coverage was designed to reveal the efficiency of software testing. Some empirical studies [Malaiya94] correlated test coverage and the number of defects in the software. The RePS utilizes such relationship and obtains software reliability based on the number of defects obtained from the test coverage. Section 3.1.2.1 discusses the issues with the traditional definition of Test Coverage and how it was resolved in [Smidts00] and Section 3.1.2.2 provides the approach given by the authors to estimate reliability from the modified Test Coverage.

3.2.2.1 Test Coverage Modification to Take Missing Functionalities into Account

The software engineering literature [IEEE98] defines multiple test coverage measures such as block (also called statement) coverage, branch coverage and data flow coverage. Only the statement coverage was selected in [Smidts00]. Statement coverage is defined as [IEEE98]:

$$\text{Statement Coverage} = \frac{LOC_{Tested}}{LOC_{Total}} \qquad (3\text{-}2)$$

where

LOC_{Tested} number of lines of code implemented that are being executed by the test data documented in the test plan [Lockheed98C].

LOC_{Total} total number of lines of code [Lockheed98B].

However, (3-2) does not take the unimplemented functions specified in the requirements into consideration. Since these functions were not implemented, the portion of code these would have constituted had they been implemented is unknown. An equivalent line of code count for these unimplemented functionalities was calculated by: 1) counting the number of function points corresponding to the missing functionalities, 2) using documented backfiring rules [Jones96] to compute an equivalent line of code count for the missing functionalities.

Therefore, (3-2) was modified in [Smidts00] to take the missing functionalities into account. This yielded:

$$\text{Statement Coverage} = \frac{LOC_{Tested} + LOC_{Miss}}{LOC_{IMPL} + LOC_{Miss}} \qquad (3\text{-}3)$$

where

LOC_{Miss} The number of lines of code for the missing functionalities

LOC_{IMPL} The number of lines of code implemented

Now using the backfiring rule (the number of lines of code of software is empirically proportional to the number of function points),

$$LOC = k * FP \qquad (3\text{-}4)$$

where

LOC The number of lines of code in the software

k The backfiring coefficient, dependent on the specific programming language used

FP The number of function points contained in the software

So now (3-3) was written as

$$\text{Statement Coverage} = \frac{LOC_{Tested} + k * FP_{Miss}}{LOC_{IMPL} + k * FP_{Miss}} \qquad (3\text{-}5)$$

where

FP_{Miss} The number of function points corresponding to the missing functionalities in the requirements specifications

The backfiring coefficient for C++ is available in the public data domain, and ranges from 40 to 140 (mode 55) LOC/FP in [Jones96].

3.2.2.2 Reliability Estimation from the Modified Test Coverage

Given the modified value of test coverage defined in Section 3.1.2.1, the number of defects remaining in the software, N, was estimated using Malaiya's results [Malaiya94, Malaiya98] which can be summarized as

$$N = N^0/C^0 \qquad (3\text{-}5)$$

where N^0 The number of defects found by test cases provided in the test plan,

C^0 The defect coverage, which is defined in [Malaiya94, Malaiya98] as the fraction of defects found by test cases given in the test plan. C^0 is given as

$$C^0 = a_0 * \ln(1 + a_1(\exp(a_2 * C_1)-1) \qquad (3\text{-}6)$$

Where, a_0, a_1, a_2 are coefficients and C_1 is the *s*tatement coverage defined in Section 3.1.2.1. The coefficients can be estimated from field data [Malaiya94, Malaiya98]. Hence knowing C_1 one obtains C^0 and knowing N^0, N is obtained.

Now, the number of defects remaining that contributes to failure is $(N - N_{repaired})$ where $N_{repaired}$ is the number of defects fixed from among N^0 , however, we do not have information about what these unknown defects are and where they are located. Therefore an approximation is made to find the probability of failure per demand, p_f: i.e.

$$p_f = 1 - e^{-\nu_K (N - N_{repaired})} \approx \nu_K \times (N - N_{repaired}) \qquad (3\text{-}7)$$

where ν_K is an average value that can be estimated from the known failure probability and the number of defects remaining in the software[Smidts00]. For instance

$$\nu_K = \frac{p_f^0}{N^0 - N_{repaired}} \qquad (3\text{-}8)$$

where p_f^0 the failure probability caused by the number of known defects remaining, N^0- $N_{repaired}$

3.2.3 Requirements Traceability RePS

In this section the approach in [Smidts00] of constructing a RePS from the Requirements Traceability measure is described.

The measure "Requirements Traceability" is defined in [IEEE98] as:

$$RT = \frac{R1}{R2} \times 100\% \qquad (3\text{-}9)$$

where

RT The value of the measure "Requirements Traceability"

R1 The number of requirements implemented in the source code

R2 The number of final requirements. We know that R2 is a function of time as requirements gets added or deleted during the software life cycle. However the model used here assumes that R2 is the final set of requirements.

This definition requires the count of R1 and R2. Unfortunately, [IEEE98] does not provide rules to perform such counting. [Smidts00] thus utilized the concept of Master Requirements Lists (MRLs) [Lockheed98A] to decompose the requirements specifications.

Each MRL can be further decomposed into a number of verbs or verb phrases that represent end-user meaningful requirements primitives. For instance, the requirements "Any failure of the system shall default to a 'Access Denied' message to the reader and a message to the attending guard 'System Failure.' The system shall default to a locked-gate with guard override capability" can be decomposed into the following four MRLs:

MRL1: Any failure of the system shall default to a 'Access Denied' message to the reader

MRL2: and a message to the attending guard 'System Failure.'

MRL3: The system shall default to a locked-gate

MRL4: with guard override capability.

Quantities *R1* and *R2* are then counted at this primitive level. Each unimplemented function was considered as a defect. Any functionality not defined in the requirements and implemented was also considered a defect.

Once the set of defects is identified, the EFSM technique is then used to calculate the failure probability propagating this specific set of defects.

3.2.4 Function Point RePS

Function point is designed to determine the functional size of the software. This measure can be determined at any stage of the software life cycle starting from the requirements specification phase as a basis to assess software quality, costs, documentation and productivity. Function points have gained acceptance as a primary measure of software size. Function points measure the size of an entire application as well as that of software enhancements, regardless of the technology used for development and/or maintenance.

Jones summarized the state-of-the-practice of the U.S. averages for delivered defects in [Jones96]. Table 3.46 in [Jones96] provides the average numbers of delivered defects per function point for different types of software systems (end-user software, management information systems, outsourced and contract software, commercial software, system software, and military software). The number of delivered defects can be obtained using the table by interpolation. Since the *a priori* knowledge of the defects' type and location and their impact on failure probability is not available, the EFSM technique cannot be applied to quantify the failure likelihood. Therefore the traditional relationship below was applied:

$$p_s = e^{-\frac{K}{T_L}N\tau} \quad (3\text{-}10)$$

Where

p_s is the probability of success per demand. A demand is an execution of the software representing its usage.

K fault exposure ratio, the average value is 4.2 x 10^{-7} failure/fault in [Musa87, Musa98]

T_L linear execution time.

N number of defects

τ average execution time per demand

τ is obtained by analyzing the reliability testing data (total testing time divided by number of test cases). In order to estimate the linear execution time, a piece of linear code is created using the same language as the original application. The statements in this simulated code follow the same pattern as the application. By pattern, it means the coding style and frequency at which a type of statement appears. The simulated code is at best an approximation of the actual code [Li06]. The simulated code is executed multiple times and the average execution time per run is obtained.

3.2.5 Bugs per Line of Code RePS

[Smidts00] also constructed a RePS to estimate failure probability from the bugs per line of code metric. Gaffney [Gaffney84] established that the number of defects remaining in the software (*N*) could be expressed empirically as a function of the number of line of codes. That is,

$$N = \sum_{i=1}^{M} (4.2 + 0.0015 S_i^{4/3}) \qquad (3\text{-}11)$$

where

i The module index

M The number of modules

S_i The number of lines of code for the ith module.

A module is defined as "an independent piece of code with a well-defined interface to the rest of the product" [Scach93], and since this definition is satisfied by the notion of class, we can substitute the idea of "module as a subroutine" to the idea of "module as a class".

The reliability estimation from this measure follows (3-10). The parameters T_L and τ are obtained in the same manner as mentioned in Section 3.1.4.

3.3 Summary

This chapter discussed the RePSs in details. The construction of RePSs is an analytical approach that links measures to defects and then defects to reliability estimation. The rationale behind the construction of RePSs provides the basis of construction of error models for the simulation. The next chapter illustrates the construction of error models for the simulation in details.

Chapter 4 Simulation and Simulation Results

In this chapter the theory behind the simulation that was carried out to determine the nature of the errors/correction factors is provided. Subsequently the results of the simulation are also presented.

4.1 Theory behind Simulation

Simulations were carried out to determine the nature of the errors for a variety of reasons. Traditionally the nature of the errors is determined experimentally which includes the extensive and difficult task of designing the experiment in a way to counter threats to validity, and executing it. Also each experiment needs a minimum number of data points (the larger the number, the better) in order to statistically validate it. This is expensive not only from a cost perspective but also from a time perspective. Sometimes carrying out an experiment is just not feasible due to lack of resources. Simulation is an alternative which provides a solution to the above problems. Moreover, simulation allows us to use a wide range of inputs. This not only provides a broader spectrum of possibilities but also acts as a catalyst for sensitivity analysis of the inputs/values. In this section, the rationale behind the simulation of each of the error forms is provided.

However, in order to simulate the error, which is defined as the difference between the real failure probability and the failure probability predicted by the model, the real failure probability requires to be modeled. As again, we do not know the reality in its totality. Therefore our goal is to model the real failure probability as it deems appropriate in the context of the models and the model uncertainty

framework. From the model uncertainty framework perspective, the experimental value is assumed to be the real value and is compared against the value predicted by the model. Therefore we simulate the experimental process to obtain the real reliability. The experimental failure probability is obtained through reliability testing. The steps are:

1) Construction of an EFSM [Wang93, Li06] representing the user's requirements and embedding user's operational profile information. Testmaster tool [Testmaster99] was used for this purpose.

2) Execution of the model to evaluate the impact of the defects. A large number of test cases are run through the application and the ratio of number of test cases failed over the total number of test cases run, gives the real failure probability. Test cases were generated from the models using *Winrunner* [Winrunner01].

As we can see, the total number of defects and their impact provide the key to the real failure probability. In order to simulate the real failure probability, the number of defects and their impacts require to be determined.

The number of defects is obtained from [Jones96], Table 3.46, which is US averages for delivered defects per function point. Jones's [Jones96] analysis is based on more than 6700 software projects.

The fault exposure probability of each defect is considered in order to assess the impact of these defects. Fault exposure probability is defined as the probability that a fault leads to failure. Therefore if there are N defects remaining in an application the real failure probability of the application is the sum of the fault

exposure probability, *k*, of each of these, i.e., the real probability of failure is $\sum_{i=1}^{i=N} k_i$, where k_i is the fault exposure probability of defect *i.* The assumption here is that the defects are mutually exclusive of each other. This is a fair enough assumption as it only excludes the case in which one fault masks others [Wu93]. As long as this situation does not happen frequently, the mutually exclusive faults assumption will be a fairly good approximation [Wu93].

Moreover, mutual exclusiveness of defects assumption gives the maximum failure probability. This is because it assumes that the defects are on different paths of the program and contribute independently to the failure probability. So, if there is interaction among defects, which arises when a defect masks another defect, the failure caused by the masked defect either will not appear in the execution of the program [Wu93] or will appear partially. Therefore the assumption of mutual exclusiveness gives a conservative value for the failure probability.

Most importantly, in this study we are concerned with the error rather than the absolute value of the real failure probability. Section 4.1.1 through Section 4.1.5 discusses the construction of error models in great details but they can be summarized as follows. The error

$e = \sum_{i=1}^{i=N} k_i - \sum_{i=1}^{i=N_1} k_i^*$ i.e. the difference between the real failure probability given by $\sum_{i=1}^{i=N} k_i$ and predicted value of failure probability given by $\sum_{i=1}^{i=N1} k*_i$. N is actual number of defects present in the application and N1 is the number of defects detected. *k* and *k** are similar and the difference in *k* and *k** may arise due to the modeler's

subjective understanding of the system which may lead to mapping of the defects at different spots/levels of the EFSM [Wang93, Li06]. This may happen in very large and complex systems and can lead to either overestimation or underestimation of errors. Since the assumption of mutual exclusiveness is also extended to the modeling of the predicted failure probability, the overestimation of failure probability cancels out to a large extent.

Figure 4-1 shows the real error vs. the modeled error. Since both the real and the modeled failure probabilities are overestimated, it is reasonable to believe that the real error is similar to the modeled error. However this will depend on the amount of overestimation made for the real failure probability and the estimated failure probability. If there are N defects actually residing in the application and all the defects are detected, the overestimation made for the real failure probability and the estimated failure probability are the same. However as the number of defects detected decreases, the difference between the overestimation for the estimated failure probability and the overestimation for the real failure probability increases. This is because of the way the error is modeled (eq(4-1)). This similarity of real and modeled errors can be studied further and is an avenue of future research. Appendix C provides further investigation on the mutual exclusiveness of defects and possible directions for future research.

An experiment was conducted to determine the similarity between real and simulated errors. The experimental results presented in Section 5.6.3 statistically show that there is not enough evidence to reject the hypothesis that the simulated errors are similar to the real errors.

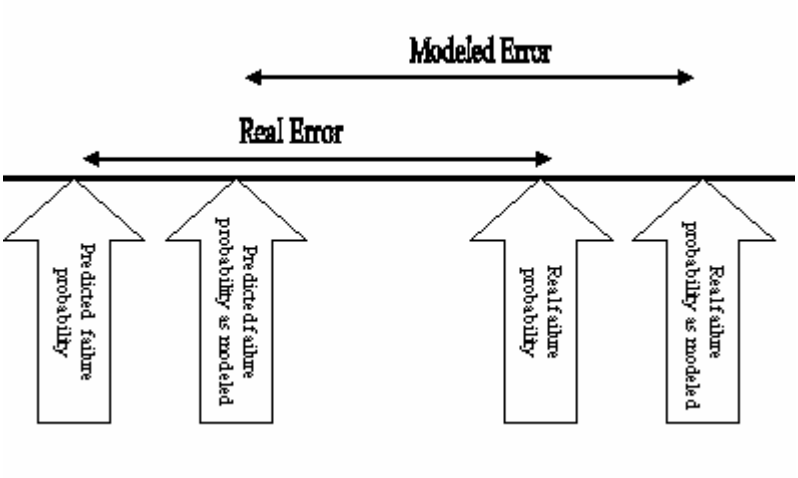


Figure 4-1 Real error vs. modeled error

The following examples illustrate that assumption of mutual exclusiveness provides a conservative value for the failure probability. The "real" versus the "modeled" failure probabilities are also discussed.

Case1:

Let us consider a program structure as shown in the following flow diagram

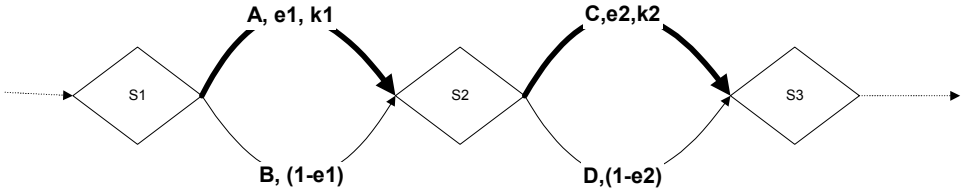


Figure 4-2 Control flow graph -1

Say S1, S2, S3 are predicates and A, B, C, D, are the different paths taken. The shaded paths A and C contain a defect each with a fault exposure probability of *k1* and *k2*.

i.e $k1 = e1 \times I1 \times P1_{End}$ where $e1$ is the execution probability of path A, $I1$ is the infection probability of the defect in path A and $P1_{End}$ is the probability that the defect propagates to the end. Similarly,

$k2 = e2 \times I2 \times P2_{End}$ where $e2$ is the execution probability of path B, $I2$ is the infection probability of the defect in path B and $P2_{End}$ is the probability that it propagates to the end.

Thus the real failure probability is equal to $k1 + k2 - k1 \times k2$. However the modeled failure probability is $k1 + k2$ and is thus a conservative estimate.

In this case the if both the defects are detected, the actual estimated failure probability is $k1^* + k2^* - k1^* \times k2^*$ and the estimated failure probability as modeled is $k1^* + k2^*$. Therefore the modeled error $(k1 + k2 - k1^* + k2^*)$ on an average is equal to zero. The actual error $(k1 + k2 - k1 \times k2 - (k1^* + k2^* - k1^* \times k2^*))$ on an average is also equal to zero. If one of the defects is detected the actual error is $(k1 + k2 - k1 \times k2 - k1^*)$, where as the modeled error is $(k1 + k2 - k1^*)$, which is also a conservative estimate. Moreover since fault exposure probability values are less than one, their multiplicative values are small and tend towards zero.

Case2:

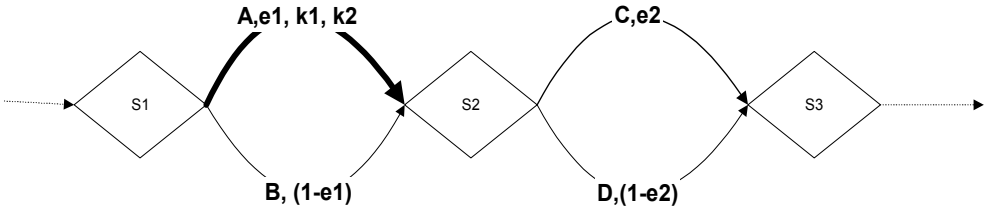


Figure 4-3 Control flow graph - 2

Let us assume that there are two defects on the path A, each with a fault exposure probability *k1 and k2*. Now, the effect of the masked defect may not show at all. For example in the above case, the real failure probability is equal to *k1*. However the modeled failure probability is *k1+k2* and is also a conservative estimate. The table below provides an example where the effect of the masked defect may not show at all.

Correct Code	Incorrect Code
…….	…….
if(x !=1)	if(x !=1)
{	{
y = 4× (x-1);	y = 4× (x-2);
z = y/6;	z = y/3;
}	}
else	else
{	{
……...	……...

Table 4-1 Code that shows that the masked defect may not have any effect

As can be seen there are two defects in the incorrect code and both are in the same path. Now, if x equals 2, y in the case of incorrect code will always be equal to 0 and the value of z is also always equal to zero. Here the defect in the statement "z = y/3" does not affect the failure probability.

In the above case with two defects on the path A (Figure 4-3), the defects may cancel each other in which case the real failure probability is zero whereas the modeled failure probability is *k1+k2* and is also a conservative estimate. The table below provides an example where the defects may cancel each other.

Correct Code	Incorrect Code
…….	…….
if(x < 10)	if(x !=1)
{	{
y = x+1;	y = x+2;
z = y+4;	z = y+2;
w = y+z;	w = y+z;
}	}
else	else
{	{
……..	……..

Table 4-2 Code that shows two defects may cancel each other

As can be seen from the above table, in case of the correct code,

w = y+z = y+y+4 = 2(x+1)+4 = 2x+6;

and in the case of incorrect code,

w = y+z = y+y+2 = 2(x+2)+2 = 2x+6. Hence the defects cancel each other.

Now consider Figure 4-4. Paths A, C and D contain a defect each. Now, the real failure probability is equal to $k1 + k2 + k3 - k1 \times k2 - k1 \times k3$. Please note that $e2 + e3 + e4$ is equal to one. The modeled failure probability is $k1 + k2 + k3$ which is a conservative estimate.

In this case, real estimated failure probability if all three defects are found is $k1^* + k2^* + k3^* - k1^* \times k2^* - k1^* \times k3^*$. Here also the error remains the same i.e. equal to zero. If two of the three defects are detected, the real estimated failure probability is $k1^* + k2^* - k1^* \times k2^*$. Therefore the real error is $(k1 + k2 + k3 - k1 \times k2 - k1 \times k3 - (k1^* + k2^* - k1^* \times k2^*))$ whereas the modeled error is $(k1 + k2 + k3 - (k1^* + k2^*))$.

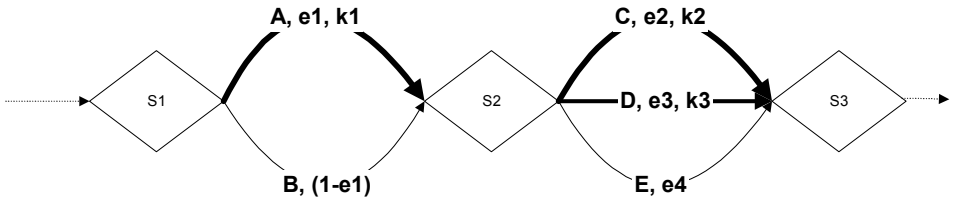


Figure 4-4 Control flow graph – 3

Now consider the following program structure. In this case the predicate S1 has a defect with a fault exposure probability *k1* and path A has a defect with fault exposure probability *k2*. However the failure probability will be less than or equal to *k1+ k2*. This is because if the defect in S1 does not affect the path A at all, the two defects together will be mutually exclusive of each other and the total failure probability will be *k1 + k2*. However, if the defect in S1 also affects the path A, the total failure probability will be less than *k1+ k2*.

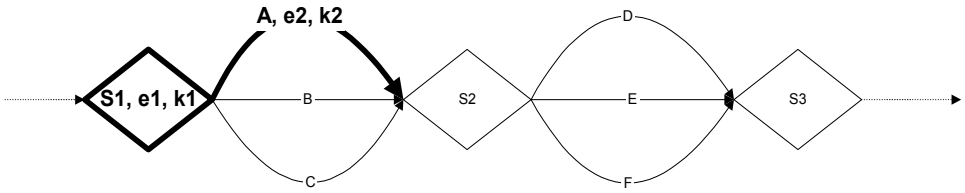


Figure 4-5 Control flow graph – 4

In the same manner it can be shown that the mutual exclusiveness assumption always provides a conservative estimate. Moreover the modeled errors are also similar to the real errors. In fact, they are equal to zero if all the defects are detected. On an average (with a data set of 6700 software projects from various domains, languages and sizes) 80% of the defects are detected [Jones96], and also since the fault exposure probabilities are less than one (the multiplicative values of

the fault exposure probabilities approach to zero), the assumption that the real and the modeled errors are similar is reasonable for all practical purposes. In fact, Capers Jones, in one of his latest talks on the state of the art of software quality in 2005 (http://www.umsec.umn.edu/files/SQA05l.pdf), says that defect removal efficiency, on an average, is 85%. However, as mentioned before, the similarity of real and modeled errors can be studied further and is an avenue of future research.

4.1.1 Defect Density Error Model Simulation:

The rationale behind the error form for Defect Density is simple. The predicted failure probability is the sum of the fault exposure probabilities of the number of defects that were identified through inspection, say N1. [Jones96] suggests that on an average N1 is equal to 0.8N. Therefore the error term is the difference between the real failure probability given by $\sum_{i=1}^{i=N} k_i$ and predicted value of failure probability given by $\sum_{i=1}^{i=N1} k*_i$ i.e.

The error form for defect density is given by

$$e = \sum_{i=1}^{i=N} k_i - \sum_{i=1}^{i=N_1} k_i^* \qquad (4\text{-}1)$$

where

- e denotes the error
- k is the real value of the fault exposure probability
    - k has a lower bound of 10^{-24} . This is because in safety critical applications, failure probabilities in the order of 10^{-9} are not unheard

of. Butler [Butler93] mentions the fact that ultra-reliable safety critical systems are required to have failure probabilities of 10^{-7}-10^{-9} per hour. He also mentions that it is not unusual to find "iteration rates" of 10-100 cycles per second. Considering 100 cycles per second, a failure rate of the order of 10^{-12}-10^{-14} per demand [Butler93] is obtained. 10^{-24} is taken to be the lower bound as a conservative estimate. Any evidence of a failure probability lower than that does not exist.

- The upper bound of *k* is varied in order to simulate a population of software. This would give us a better picture with a range of failure probabilities. The upper bound of *k* is set at 10^{-1}, $10^{-2,}$ $10^{-3,}$ $10^{-4,}$ $10^{-5,}$ 10^{-6} for different sets of simulations. This set covers the lower and the moderately reliable applications [Butler93]. Moreover, the characteristics of the error did not change and were predictable for lower order of failure probabilities. However, an upper bound of 10^{-13} is also considered to represent ultra-reliable applications.
- *k* is log-normally distributed within the above range (which is also taken as the six sigma range) . Various studies suggest that software failure rate is log-normally distributed [Mullen98A, Mullen98B]. The main reasoning is that since event rates in software systems are generated by multiplicative processes there is reason to believe that the distribution of rates of events, including, failure rates, is lognormal. Also many previously published empirical studies show

that the failure rate distributions are well fit by the lognormal probability distribution [Mullen98A, Mullen98B].

- k^* is the predicted value of fault exposure probability.

  Models, as we know are the means by which we reflect and express our understanding of some aspect of reality, a particular unknown of interest. Even though the difference between models and reality at a fundamental level always exists, the issue is whether the prediction or performance of the model is correct at some practically acceptable level, within the model's domain of application. [Musa87] studied the performance of various software reliability models and found them to be fairly good approximations of reality. Intuitively, we can say that k^* should be of the same nature as k and close to k. Moreover the difference in k and k^* may arise due to the modeler's subjective understanding of the system which may lead to mapping of the defects at different spots/levels of the EFSM [Wang93, Li06]. This may happen in very large and complex systems and can lead to either overestimation or underestimation of errors. Hence a reasonable assumption is to take k^* as normally distributed around k ie with a mean of k.

  - The standard deviation of k^* is varied in order to simulate the different population of software and/or models. Musa *et al* [Musa87] have noticed that the error in the prediction of different software reliability models vary to a maximum of about 30%. [Malaiya92]

studied four software reliability growth models and found similar results. For the simulation purposes, the value of standard deviation is varied between *0.01k*, 0.05k*, 0.1k*, 0.15 k*, 0.20 k*, 0.25 k*, and 0.3 k* and 0.4k**.

- N is the number of defects remaining in the application and is obtained from [Jones96], Table 3.46, which is US averages for delivered defects per function point. We wanted to simulate different sizes of software applications. Therefore functional size is also varied and the different functional sizes taken into account are 75, 150, 300, 600, 1200 and 2400 and 10,000 functional points. As per Jones [Jones96], one function point refers to 128 lines of code in C. Therefore 10,000 function points refer to more than a million lines of code in C.
- N1 is the number of defects found and removed. As per Jones [Jones96], the defect removal efficiency on an average is 80%. Therefore N1 is equal to 0.8*N. However for the simulation purposes, N1 is varied from 0.1*N to 0.8*N with an interval of 0.1. N1 is selected uniformly from among N. This means that every defect has equal probability of getting detected irrespective of its fault exposure probability. In the vast literature, software inspection and reviews have been studied extensively over the years. Primarily these studies have been to understand the efficacy of software inspection and the events or sources that help detecting a defect. There is no evidence that fault exposure probability influences the detectability of a defect. [Porter98] studied the sources of variation in software inspections and found that the

inspection experience, language familiarity and application experience of the reviewers; the language familiarity and application experience of the code author, the type of change, functionality, code structure, code size and pre-inspection testing of the code units are the main sources of variation of the efficacy of inspection. Pre-inspection testing refers to unit testing performed by developers before the inspection. Another significant research work is presented in [Chaar93] where the authors studied the events that helps detect such a defect. [Anda02], [Porter95A], [Porter95B], [Kelly03], [Kelly00], [Dunsmore01], [Laitenberger99] and numerous other studies on software inspection do not consider fault exposure probability at all.

The observation that software defect detection during software inspection is independent of the fault exposure probability of the defect can be justified because before or during inspection, the fault exposure probabilities of the defects are usually never known. Moreover the inspection techniques (ad-hoc, checklist based or scenario based [Porter95A]) do not require the knowledge of fault exposure probabilities of the defects. Therefore it can be assumed that the detectability of the defects is independent of the fault exposure probability of a defect.

However, if the historical data of a certain organization hints at a relationship between the detectability of a defect and its fault exposure probability, such a relationship can be explored and established and then incorporated into the error model.

4.1.2 Test Coverage Error Model Simulation

Let the number of defects before the testing process begins be $N_{before\_test.}$ Let the number of defects detected through testing be N_{test}. The defects are then repaired and if ρ is the defect repair probability, the number of defects repaired,

$$N_{repaired} = \rho * N_{test.} \quad (4\text{-}2)$$

Now the number of defects which are either not repaired or are the ones which when repaired lead to addition of new faults is equal to $(1\text{-}\rho)* N_{test.}$ If the number of defects added due to bad fixes is $N_{bad\_fix.}$, the true number of defects remaining after test,

$$N_{after\_test} = N_{before\_test} - N_{repaired} + N_{bad\_fix} \quad (4\text{-}3)$$

Therefore the real probability of failure is $\sum_{i=1}^{N_{before\_test}-N_{repaired}} k_i + \sum_{i=1}^{N_{bad\_fix}} k'_i$. Here the ks' refer to the fault exposure probabilities and are sampled from among the fault exposure probabilities already selected for the defects, N_{before_test}, and k' refers to new fault exposure probability that exists due to addition of new faults.

In order to predict the failure probability, the number of defects remaining in the code is considered as always. The number of defects estimated to be in the code is N_{test}/C^0 where C^0 is the defect coverage .The defect coverage is defined in [Malaiya94, Malaiya98][Jones96] as the fraction of defects found by test cases given in the test plan. In the RePS construction from the Test Coverage measure, a perfect repair probability was considered. Hence the number of defects repaired is N_{test}. Therefore the number of defects remaining that contributes to failure is $N_{test}/C^o - N_{test}$.

Thus the estimated failure probability is

$$\frac{\sum_{i=1}^{N_{Test}} k_i^*}{N_{Test}} \times \left(\frac{N_{Test}}{C^o} - N_{Test} \right) \qquad (4\text{-}4)$$

where $\frac{\sum_{i=1}^{N_{Test}} k_i^*}{N_{Test}}$ is the average value of fault exposure probability per fault. This corresponds to ν_K in Section 3.1.2.2.

Therefore the error form for Test coverage is given by

$$e = \sum_{i=1}^{N_{before\_test} - N_{repaired}} k_i + \sum_{i=1}^{N_{bad\_fix}} k'_i - \frac{\sum_{i=1}^{N_{Test}} k_i^*}{N_{Test}} \times \left(\frac{N_{Test}}{C^o} - N_{Test} \right) \qquad (4\text{-}5)$$

where

- e denotes the error
- The first two terms of (4-5) denotes the real failure probability. The fault exposure probabilities, k *and* k' are simulated in the same manner as mentioned before in Section 4.1.1.
- As per Jones [Jones96] the defect removal efficiency of a formal testing process by itself is 53% (median value). Therefore N_{test} is equal to 0.53* N_{before_test}. However since the upper and lower bound for defect removal efficiency of a formal testing process by itself is 60% and 37% respectively, simulations were conducted for these values also.
    - N_{before_test} is the number of defects remaining in the application and is also obtained from [Jones96]
    - The functional size is varied as before

- N_{after_test} is the true number of defects remaining after test and is equal to $N_{before\_test} - \rho * N_{test} + N_{bad\_fix}$
    - As far as the defect repair probability is concerned, Jones states [Jones96] that the defect removal efficiency is 85%. This value takes into account the bad fixes defects as well. Bad fixes defects are the defects that are accidentally injected while fixing an existing defect [Jones97]. The survey presented at www.softwaremetrics.com suggests a similar value. For the simulation, a ρ value of 0.62, 0.85, and 0.96, which are the lower bound, median and the upper bound values, are considered.
    - According to [Jones96, Musa87] bad fix defects are about 5% of all defects. Therefore, N_{bad_fix} will approximately be equal to $0.05*N_{test}$. In fact, Capers Jones, in one of his latest talks on the state of the art of software quality in 2005 (http://www.umsec.umn.edu/files/SQA05l.pdf), says that bad fix defects on an average is 8%. Therefore simulation for bad fix defects was conducted at 5% and 8%. Moreover, according to [Jones96, Table5.3], 0% of all the Severity-1 (most critical, system or program is inoperable) failures are caused by bad fix errors. [Sullivan91] also suggests that bug fix errors have little impact on the system availability. The authors have studied five years of field data on software defects to develop a taxonomy of defects, providing insight into their behaviour and impact. The data comes from IBM's field

service database called RETAIN. Based on these evidences, we believe that the bug fix defects will not have fault exposure probabilities higher than the fault exposure probability initially considered to be the upper bound.

- k^* is the predicted value of fault exposure probability and with the same reasoning as presented in Section 4.1.1, is normally distributed around k i.e. with a mean of k. The standard deviation of k^* is varied between $0.1*k$ to $0.4*k$ as before. Here the ks' refer to the real fault exposure probabilities and are sampled from the fault exposure probabilities already selected for the defects, N_{before_test}.
- As mentioned before, C^0 is the defect coverage and can range from 0.0 to 1.0 depending on the efficacy of the test cases. For the simulation, different values of C^0 from 0.1 to 1.0 with an interval of 0.1 are considered.

4.1.3 Requirements Traceability Error Model Simulation

Here again the logic is similar to that of the Defect Density error form simulation. If there are N defects remaining in an application, the real failure probability of the application is the sum of the fault exposure probability, k, of each of the defects. Therefore the real probability of failure is $\sum_{i=1}^{i=N} k_i$ with the same assumptions as in Section 4.1.1. The predicted failure probability is the sum of the fault exposure probability of the number of defects that were found through Requirements Traceability, say N1. Therefore the error term is the difference

between the real failure probability given by $\sum_{i=1}^{i=N} k_i$ and the predicted value of failure probability given by $\sum_{i=1}^{i=N1} k*_i$ i.e.

$$e = \sum_{i=1}^{i=N} k_i - \sum_{i=1}^{i=N_1} k_i^* \quad (4\text{-}6)$$

where

- e denotes the error
- The first term of (4-6) i.e. $\sum_{i=1}^{i=N} k_i$ is the same as in (4-1) and denotes the real failure probability
- $k*$ is the predicted value of fault exposure probability and has the same properties as in Defect Density.
- N_1 is the number of defects detected by Requirements Traceability. Since the literature does not provide an estimate of the efficiency of requirements traceability analysis, a set of values from 1.0 to 0.8 with an interval of 0.1 is considered to represent different requirements traceability efficiencies. The higher bound is taken to be 0.8 because that is the maximum defect removal efficiency as per Jones [Jones96]. The functional size is also varied as mentioned in Section 4.1.1.

4.1.4 Function Point Error Model Simulation

The error form for Function Point is given by the difference between the real failure probability and the predicted failure probability. Here the real failure

probability is computed as before. As mentioned before, in the construction of RePS from Function Point, since the *a priori* knowledge of the defects' type and location and their impact on failure probability is not available, the EFSM technique could not be applied to quantify the failure likelihood. Therefore the traditional relationship as in (3-10) is applied. Hence to simulate this error, K, or the fault exposure ratio proposed by Musa [Musa87, Musa98] is used. Therefore

$$e = \sum_{i=1}^{i=N} k_i - \frac{\tau}{T_L} * K * N_1 \qquad (4\text{-}7)$$

where

- e denotes the error
- The first term of (4-7) ie $\sum_{i=1}^{i=N} k_i$ is same as (4-1) and denotes the real failure probability
- It is usually seen [Lockheed98] that the times taken per demand τ, and the linear time of execution of the program are usually of the same order. [Malaiya93] also suggests same order of average time per execution and linear execution time. In fact the authors cite the cases where the ratio of average execution time over linear execution time may be greater than or less than one. They say that if a program is *loop dominated*, i.e. the program execution involves a large number loops, the ratio may be greater than one. For a *branch dominated* program, the ratio may be smaller than one since during a single execution, many branches would not be executed. Therefore

to represent various populations of software we vary the $\frac{\tau}{T_L}$ ratio and the values taken into account are 0.1, 0.01, 0.001, 1, 10, 100 and 1000.

- K is the fault exposure ratio proposed by Musa [Musa87, Musa98] and is $4.2 * 10^{-7}$ failure/fault.
- N_I is the number of defects estimated as a function of number of Function Points and is obtained from [Jones96]. The functional size is also varied as before.

4.1.5 Bugs per Line of Code

As mentioned before, Gaffney [Gaffney84] established that the number of defects remaining in the software (N_I) could be expressed empirically as a function of the number of line of codes. That is,

$$N_1 = \sum_{i=1}^{M} (4.2 + 0.0015 S_i^{4/3}) \tag{4-8}$$

Therefore, the predicted failure probability is given by $\frac{\tau}{T_L} * K * \sum_{i=1}^{i=M} (4.2 + 0.0015 S_i)$. Also, during the construction of RePS from Bugs per Line of Code the relationship as in (3-10) was used to determine the failure probability instead of the EFSM technique as again, the *a priori* knowledge of the defects' type and location and their impact on failure probability is not available. Therefore, the error term is given as

$$\mathrm{e} = \sum_{i=1}^{i=N} k_i - \frac{\tau}{T_L} * K * \sum_{i=1}^{i=M} (4.2 + 0.0015 S_i) \tag{4-9}$$

where

- e denotes the error
- The first term of (4-9) ie $\sum_{i=1}^{i=N} k_i$ is same as (4-1) and denotes the real failure probability
- We vary the $\frac{\tau}{T_L}$ ratio in a similar fashion as mentioned in Section 4.1.4.
- K is the fault exposure ratio proposed by Musa [Musa87, Musa98] and is $4.2*10^{-7}$ failure/fault
- The term $\sum_{i=1}^{M}(4.2+0.0015S_i^{4/3})$ gives the number of defects N_1, as established by Gaffney [Gaffney84]. Here, M is the number of modules in the application. In order to estimate M, the following steps are taken.
    - The functional size is varied as before. For each functional size, the number of lines of code for the application is calculated as k_b*FP, where k_b is the backfiring coefficient and FP is the number of function points of the application. The backfiring coefficient of the most used languages ie C, C++, FORTRAN and VB is taken into account [Jones96]
    - There is no official standard for the number of lines of code per module. Every organization has its own standard which it tries to conform to. However there is evidence [Hatton97] that when one plots defect density versus module size, the curve is U-shaped and concave upwards which means very small and very large modules

are associated with more bugs than those of intermediate size. The curve looks roughly logarithmic up to a 'sweet spot' where it flattens (corresponding to the minimum in the defect density curve), after which it goes up. Hatton's empirical results imply that the sweet spot lies between 200 and 400 lines of code that minimizes probable defect density, all other factors (such as programmer skill) being equal. This size is also independent of the language being used. Banker and Kemerer [Banker89] and Withrow [Withrow90] have also observed similar results. Based on this evidence, we vary the value of lines of code per module between 200, 300 and 400.

- The number of modules *M* is then given by

  Number of Lines of Code for the application / Number of Lines of code per module.

4.2 Simulation Results

Once the simulation for each of the error forms was designed, it was executed using the SAS statistical tool (www.sas.com). This section discusses SAS and the characteristics of statistical tests that were carried out using SAS to determine the results. It also discusses the tool that was developed to carry out the simulation. Finally it discusses the simulation results in great details.

4.2.1 Statistical tests that were carried out to determine the simulation results

SAS, originally called "Statistical Analysis Software," has generally been preferred primarily due to the power of its programming language and the acceptability of its results. SAS has developed a reputation of being powerful and full-featured statistical software that allows the user to manipulate and analyze data in many different ways. Because of its capabilities, this software package is used in many disciplines, including medical sciences, biological sciences, social sciences, and education [Chen03]. SAS has changed a lot across versions, with most of the changes catering to the business community. We used SAS version 9.1, the latest version available at this point of time, for the statistical tests.

Normality and log normality tests for the errors were conducted at an $\alpha = .05$. For the log normality tests, normality tests on the natural logarithm of error values were carried out. The values provided by Shapiro Wilk tests are the most powerful. The Shapiro-Wilk test, proposed in 1965, calculates a W statistic that tests whether a random sample, $x_1, x_2, ..., x_n$ comes from (specifically) a normal distribution . Small values of W are evidence of departure from normality and percentage points for the W statistic, obtained via Monte Carlo simulations, were reproduced by Pearson and Hartley [Pearson72]. This test has done very well in comparison studies with other goodness of fit tests. The W statistic is calculated as follows:

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \qquad (4\text{-}10)$$

where the $x_{(i)}$ are the ordered sample values ($x_{(1)}$ is the smallest) and the a_i are constants generated from the means, variances and co-variances of the order statistics of a sample of size n from a normal distribution [Pearson72]. For more information about the Shapiro-Wilk test please refer to the original Shapiro and Wilk paper [Shapiro65] and the tables in Pearson and Hartley [Pearson72].

For each set of conditions, the simulation for each error form was ran 50,000 times. By a set of conditions, we mean the specifics of the data set, i.e., a particular size of the application, upper bound of failure probability, requirements traceability efficiency (if applicable), $\frac{\tau}{T_L}$ ratio (if applicable), language in which the application is coded (if applicable) etc. For every run of simulation 25 to 50 data points[1] (error values) were generated. That is because a really good normality test in SAS requires close to 30 data points but too many data points may result in overly sensitive tests in normality [Douglass04]. The final results for each error form are given below along with the stem and leaf and box plots.

The stem and leaf plot is simply a horizontal histogram. In a stem-and-leaf plot each data point is split into a "stem" and a "leaf". The first two or three digits of the value of each data point are used as the stem and the next digits are used as the

[1] A data point is a value of the variable on which statistical tests are performed

leaf. This forms a histogram that not only provides the frequency of each class but also the actual values for each observation/data point. For example, in Figure 4-7, which is a stem and leaf plot, there are 30 data points. And the values of each data point are: 0.65, 0.5, 0.34, 0.38, 0.38 and so on.

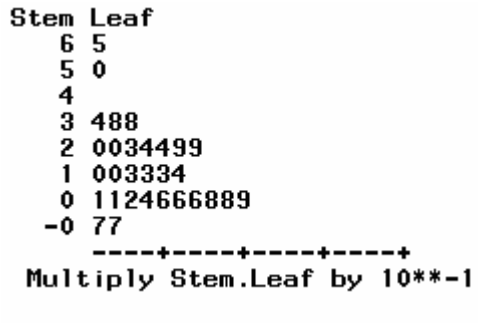


Figure 4-6 Stem and Leaf Plot

The Box Plot is an efficient method for displaying data. . The box plot is interpreted as follows:

- the box itself contains the middle 50% of the data. The upper edge (hinge) of the box indicates the 75th percentile of the data set, and the lower hinge indicates the 25th percentile.
- the bar in the center anchored with the '*' represents the median value of the data.
- the '+' in the center of the box represents the mean
- the ends of the vertical lines or "whiskers" indicate the minimum and maximum data values, unless outliers are present in which case the whiskers extend to a maximum of 1.5 times the inter-quartile range.

- The points outside the ends of the whiskers, if present, are outliers and are marked as ‘o’.

4.2.2 The Tool

As mentioned before, a tool was developed to simulate the error models. This section describes the overall framework of the tool, the type of inputs and the output generated by the system.

The tool is developed using SAS programming language. The development platform is Windows. Figure 4-7 shows the block diagram of the tool.

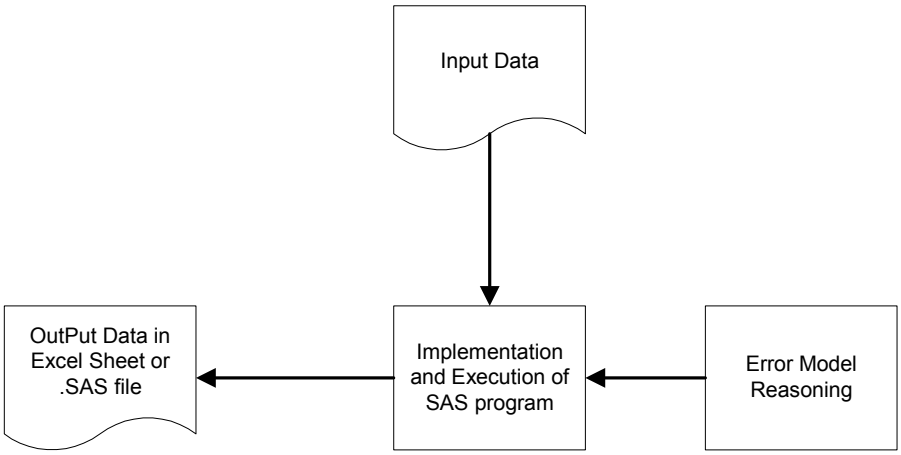


Figure 4-7 Framework of the Error simulation Tool

The input data can be incorporated into the SAS program. The SAS program along with the error model reasoning (Section 4.1.1 – Section 4.1.5) is then executed. The results can be saved in .SAS files. The tabular results can be exported into an Excel file.

The snapshot showing the incorporation of input data and a portion of the program is shown in Figure 4-8.

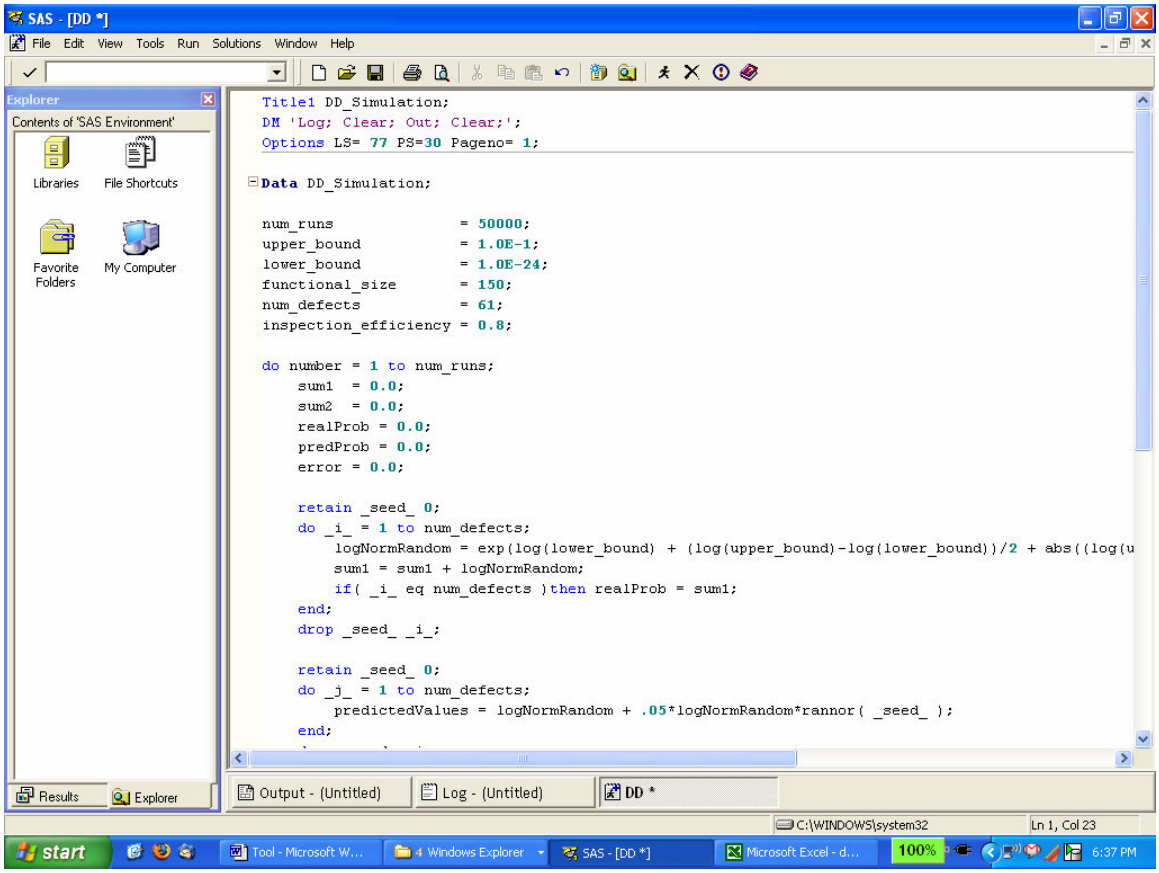


Figure 4-8 Snapshot of the program showing the incorporation of the input data

As can be seen, the input data can be incorporated at the beginning of the program. Figure 4-9 is a snapshot of the output results showing real probability of failure values, predicted probability of failure values and the error values

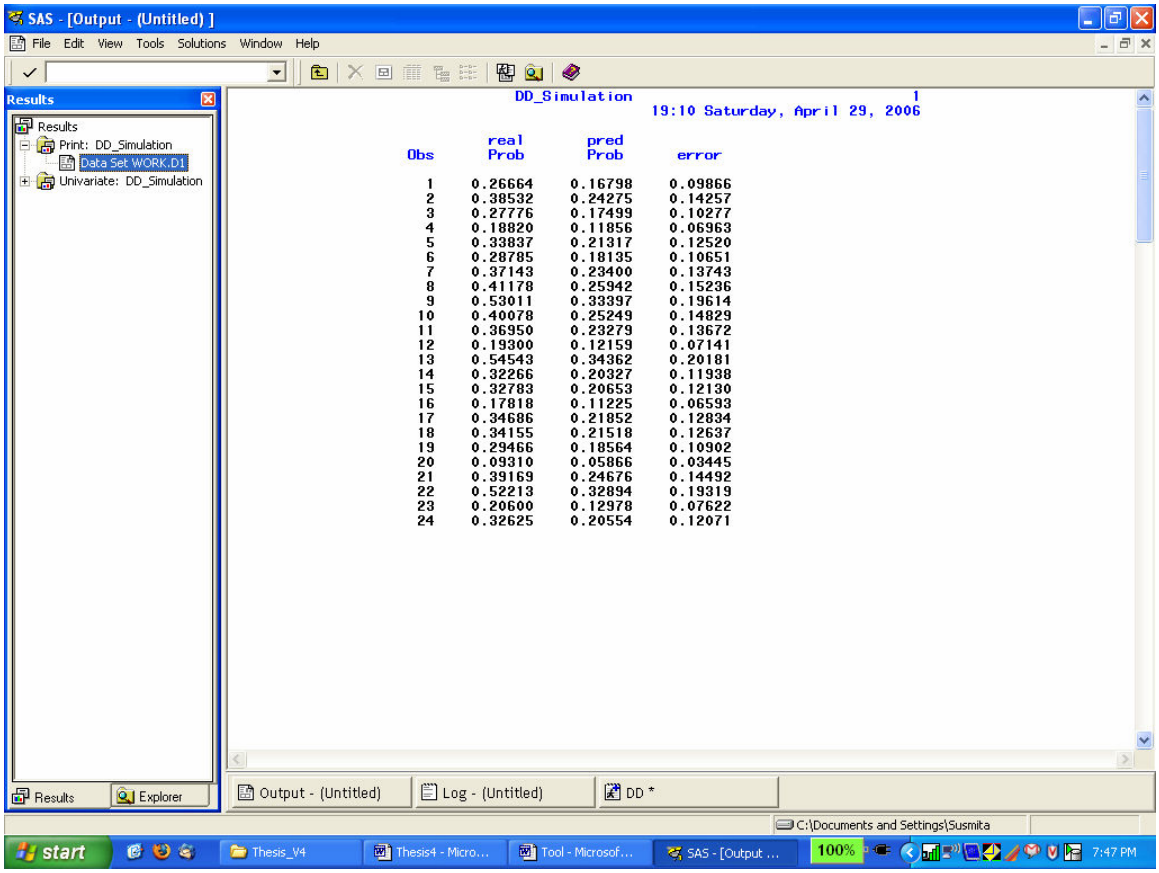


Figure 4-9 Snapshot of the results showing the probability of failure values

Figure 4-10 is a snapshot of the output results showing the mean and standard deviation of the error and Figure 4-11 shows the results of the normality tests on the error.

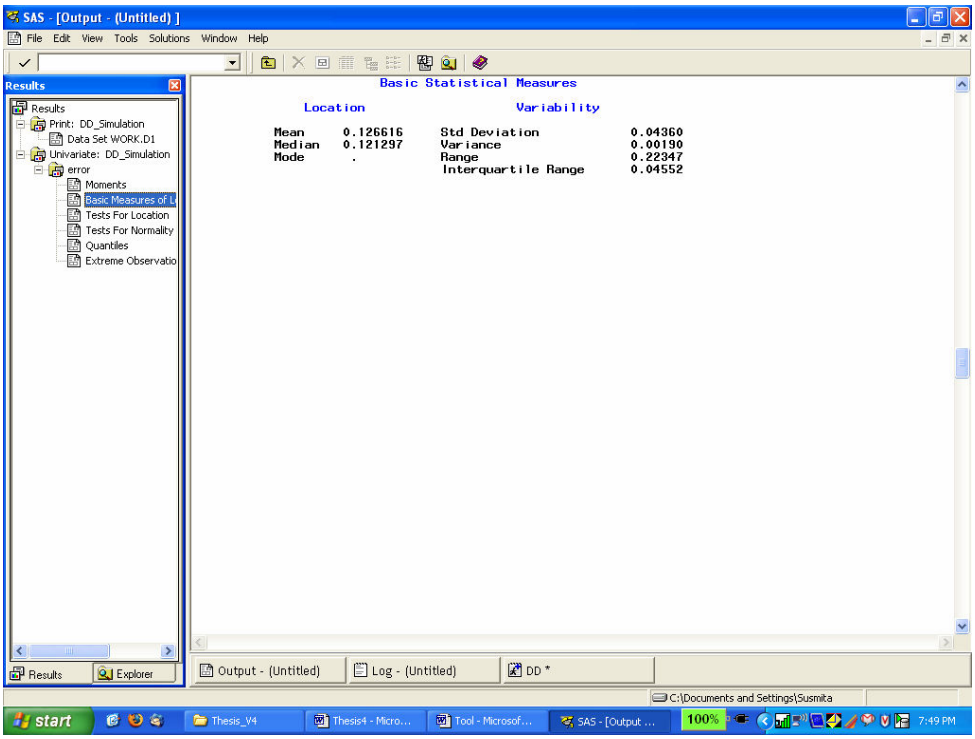


Figure 4-10 Snapshot of the results showing mean and standard deviation of the error values

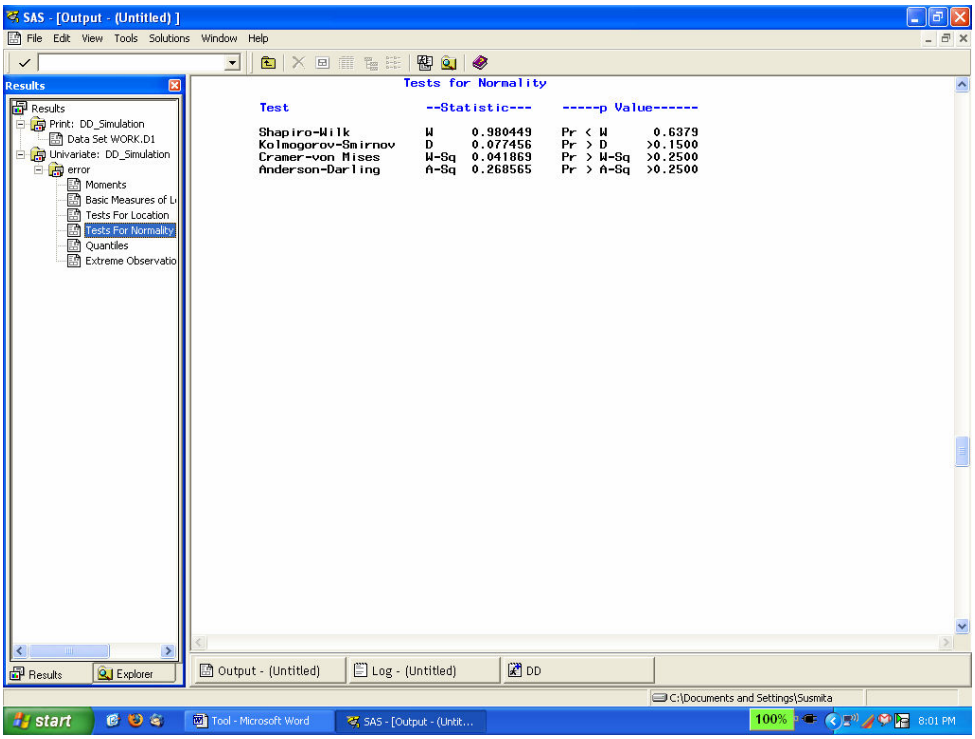
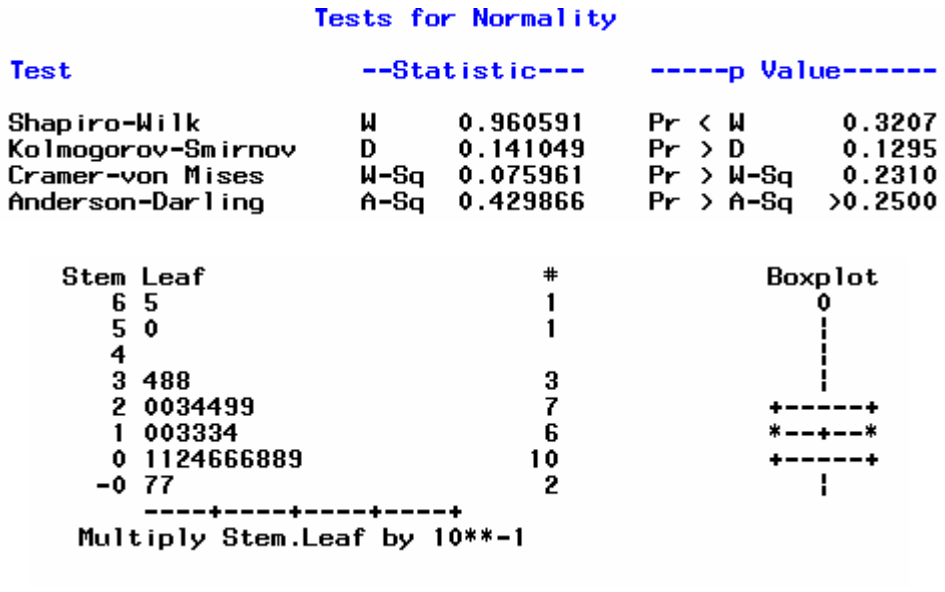


Figure 4-11 Snapshot of the results showing the tests of normality on the error values

4.2.3 Simulation Results

As can be seen, there was enough evidence to assume that the Defect Density error model follows an additive distribution. The statistics for the same are given below.



Tests for Normality

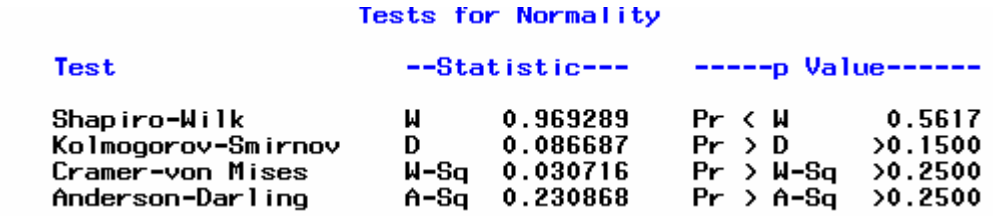
Test		--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.960591	Pr < W		0.3207
Kolmogorov-Smirnov	D	0.141049	Pr > D		0.1295
Cramer-von Mises	W-Sq	0.075961	Pr > W-Sq		0.2310
Anderson-Darling	A-Sq	0.429866	Pr > A-Sq		>0.2500

Stem	Leaf	#	Boxplot
6	5	1	0
5	0	1	|
4			|
3	488	3	|
2	0034499	7	+------+
1	003334	6	*--+--*
0	1124666889	10	+------+
-0	77	2	|

Multiply Stem.Leaf by 10**-1

Figure 4-12 Statistics for the Defect Density Error

There was enough evidence to assume that Bugs per Line of Code error model follows a multiplicative distribution. The statistics for the same are given below.



Tests for Normality

Test		--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.969289	Pr < W		0.5617
Kolmogorov-Smirnov	D	0.086687	Pr > D		>0.1500
Cramer-von Mises	W-Sq	0.030716	Pr > W-Sq		>0.2500
Anderson-Darling	A-Sq	0.230868	Pr > A-Sq		>0.2500

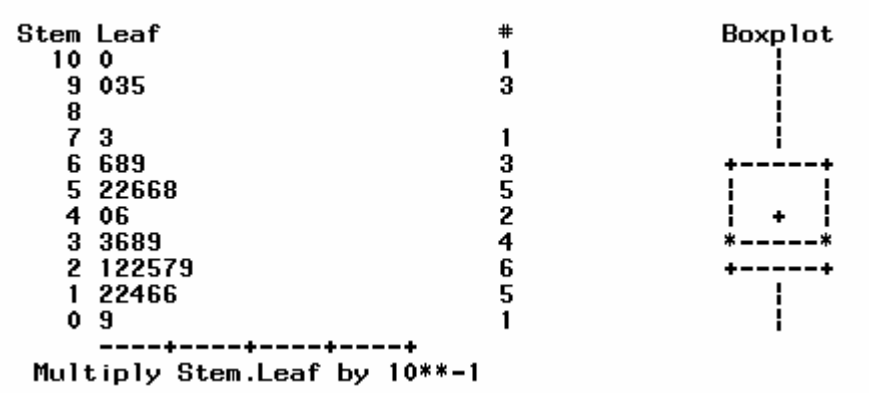


Figure 4-13 Statistics for the Bugs per Line of Code Error

There was enough evidence to assume that Function Point error model follows a multiplicative distribution. The statistics for the same are given below.

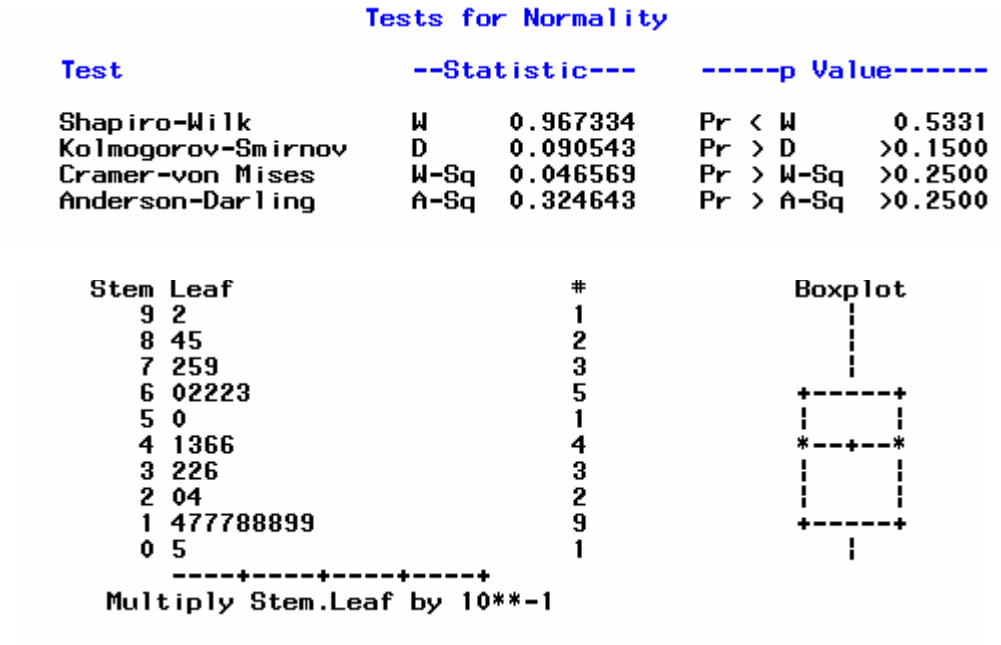
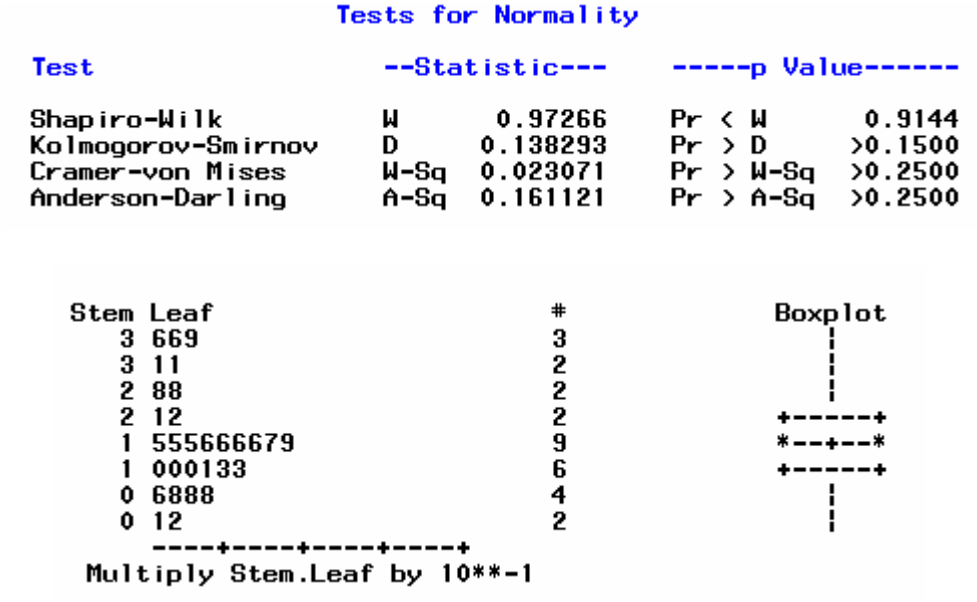


Figure 4-14 Statistics for the Function Point Error

There was enough evidence to assume that Requirements Traceability error model follows an additive distribution. The statistics for the same are given below.



Tests for Normality

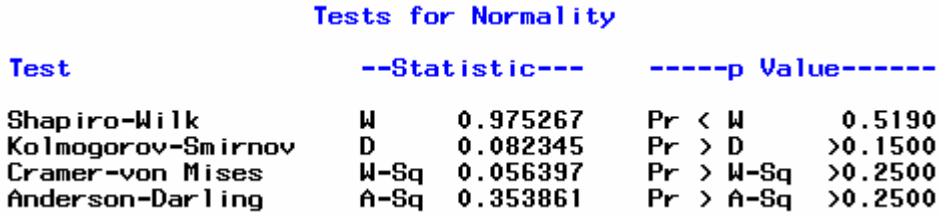
Test		--Statistic---	-----p Value------	
Shapiro-Wilk	W	0.97266	Pr < W	0.9144
Kolmogorov-Smirnov	D	0.138293	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.023071	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.161121	Pr > A-Sq	>0.2500

Stem	Leaf	#	Boxplot
3	669	3	
3	11	2	
2	88	2	
2	12	2	
1	555666679	9	
1	000133	6	
0	6888	4	
0	12	2	

Multiply Stem.Leaf by 10**-1

Figure 4-15 Statistics for the Requirements Traceability Error

There was enough evidence to assume that Test Coverage error model follows an additive error model.



Tests for Normality

Test		--Statistic---	-----p Value------	
Shapiro-Wilk	W	0.975267	Pr < W	0.5190
Kolmogorov-Smirnov	D	0.082345	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.056397	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.353861	Pr > A-Sq	>0.2500

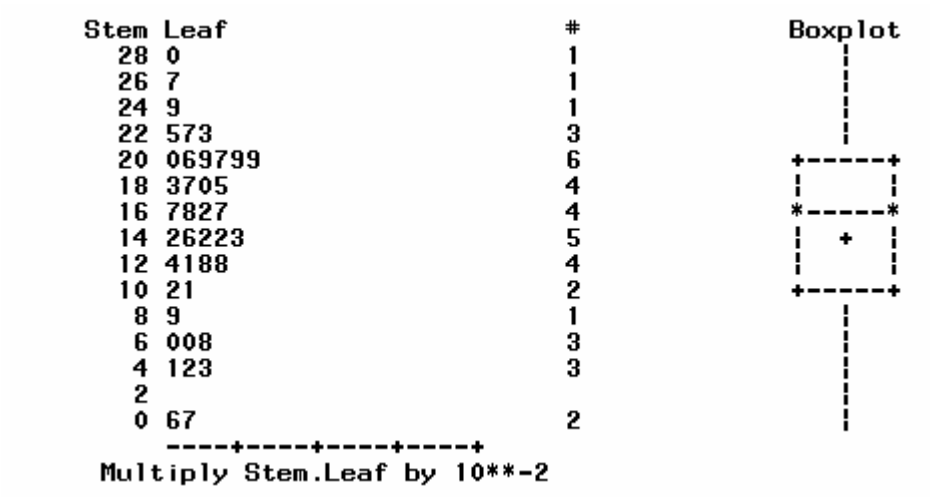


Figure 4-16 Statistics for the Test Coverage Error

4.2.4 Summary of the Simulation Results

Simulation was conducted on the different values that were considered as per the design, for example, different functional sizes, different bounds of the fault exposure probabilities, different inspection efficiencies etc. The error form did not change for any of these variations, i.e., the error form followed an additive error model for the Defect Density, Requirements Traceability, and Test Coverage errors and followed a multiplicative error model for Bugs per Line of Code and Function Point errors. In this section the mean and standard deviation of some of the results obtained for different sets of simulation is provided. The relative error percentages for some of the results are also provided. Relative error is defined as the ratio of the absolute value of the error over the real value of failure probability. More results are provided in Appendix A.

The results are then discussed in details. The change of the errors with variation of different parameters is shown through graphs.

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	.09	1.1	1.4	1.7	2.4	3.0	4.9
	Std. Dev	0.04	1.1	1.3	1.2	1.6	2.4	3.1
Requirements Traceability	Mean	0.1	1.1	1.3	1.6	2.3	2.7	4.7
	Std. Dev	0.05	1.1	1.1	1.2	1.4	2.4	2.2
Test Coverage	Mean	0.01	1.5	1.7	2	2.5	3	5.7
	Std. Dev	0.05	1	1.1	1.5	2.3	2.2	2.8
Function Point	Mean	1.1	2.1	3.8	4.0	4.4	5.4	9.6
	Std. Dev	1.9	1.6	2.5	2.6	2.3	1.8	2.6
Bugs per Line of Code	Mean	1.3	2.4	3.2	3.8	4.6	5.5	9.6
	Std. Dev	1.9	1.7	3.0	3.2	1.9	2.7	2.3

Table 4-3 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-2 (Multiply each value by 10-2)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	. 09	1.245	1.2	1.7	2.2	2.6	4.7
	Std. Dev	.04	1.3	.9	1.2	1.7	2.0	2.9
Requirements Traceability	Mean	0.10	1.2	1.4	1.8	2.3	2.7	4.7
	Std. Dev	0.04	1.3	1.1	1.3	1.9	2. 4	2.7
Test Coverage	Mean	0.12	1.3	1.6	1.9	2.4	3.0	5.6
	Std. Dev	0.06	1.2	1.1	1.7	1.8	2.0	2.8
Function Point	Mean	-9.3	-17.5	-39.4	-76.2	-255.3	-660.9	-2787
	Std. Dev	2.9	10.6	11.2	13.3	23.3	27.3	32.6
Bugs per Line of Code	Mean	-41.2	-81.3	-155.3	-315.3	-645.2	-1184	-4789
	Std. Dev	3.8	8.6	11.4	20.5	24.2	34.4	39.2

Table 4-4 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-6 (Multiply each value by 10-6)**

For a better understanding of the errors, the tables below provide the average percentages of the relative errors for varying functional sizes with upper bound of fault exposure probability of 10E-2, 10E-4 and 10E-6. Results for upper bounds of fault exposure probabilities 10E-1 are provided in Appendix A.

Functional Size Error	75	150	300	600	1200	2400	10000
Defect Density	9.29	38.36	38.56	44.54	50.45	54.09	58.45
Requirements Traceability	10.45	38.45	39.76	43.28	50.91	53.69	57.72
Test Coverage	12.16	43.34	43.87	48.43	57.27	60.83	67.38
Function Point	99.89	99.91	99.91	99.57	99.54	99.01	98.89
Bugs per Line of Code	99.86	99.87	99.45	99.43	99.32	99.03	99.00

Table 4-5 Relative error percentages for varying functional sizes with upper bound of fault exposure probability of 10E-2

Functional Size Error	75	150	300	600	1200	2400	10000
Defect Density	9.68	37.45	39.31	42.93	50.36	53.84	57.42
Requirements Traceability	9.54	38.74	39.33	43.06	49.97	53.17	59.37
Test Coverage	13.07	42.99	43.67	48.43	56.29	60.33	66.35
Function Point	58.89	67.86	94.28	77.5	62.21	22.22	185.56
Bugs per Line of Code	51.67	57.14	48.57	15.00	42.22	118.51	395.83

Table 4-6 Relative error percentages for varying functional sizes with upper bound of fault exposure probability of 10E-4

Functional Size Error	75	150	300	600	1200	2400	10000
Defect Density	9.18	37.45	38.47	44.51	48.99	53.02	60.21
Requirements Traceability	9.86	34.98	38.54	45.14	48.98	52.60	60.72
Test Coverage	13.17	44.75	46.76	49.05	56.73	59.95	70.73
Function Point	760	950	1114	1904	5667	11700	28400
Bugs per Line of Code	3400	3681	4428	8076	14300	21142	48867

Table 4-7 Relative error percentages for varying functional sizes with upper bound of fault exposure probability of 10E-6

4.3 Discussion of the Simulation Results:

Table 4-3 through Table 4-7 tells us how the errors/relative errors vary across different sizes and different failure probabilities. They also provide information on the prediction accuracies of the RePSs.

As we can see, Defect Density and Requirements Traceability errors are very close to each other. That is expected because of the similarities in the RePSs constructions from these measures. Moreover, in all the results provided above, the defect removal efficiency of Requirements Traceability is 80%, which is the same as the defect removal efficiency of Defect Density.

The relative error for defect density is around 9% for an application with a functional size of 75 function points, 37% for a functional size of 150 function points, 39% for a functional size of 300 function points, 44% for a functional size of 600 function points, 48% for a functional size of 1200 function points and 53% for a functional size of 2400 function points and 59% for a functional size of 10,000

function points. The percentages of relative errors across different orders of failure probabilities remained similar for Defect Density, Requirements Traceability and Test Coverage. Figure 4-17 shows the relative error percentages across application of varying order of failure probabilities for a functional size of 75 function points. The points in X-axis correspond to applications of different order of failure probabilities (10E-1, 10E-2, 10E-4 and 10E-6). As can be seen, the percentages of relative errors across different orders of failure probabilities remained similar.

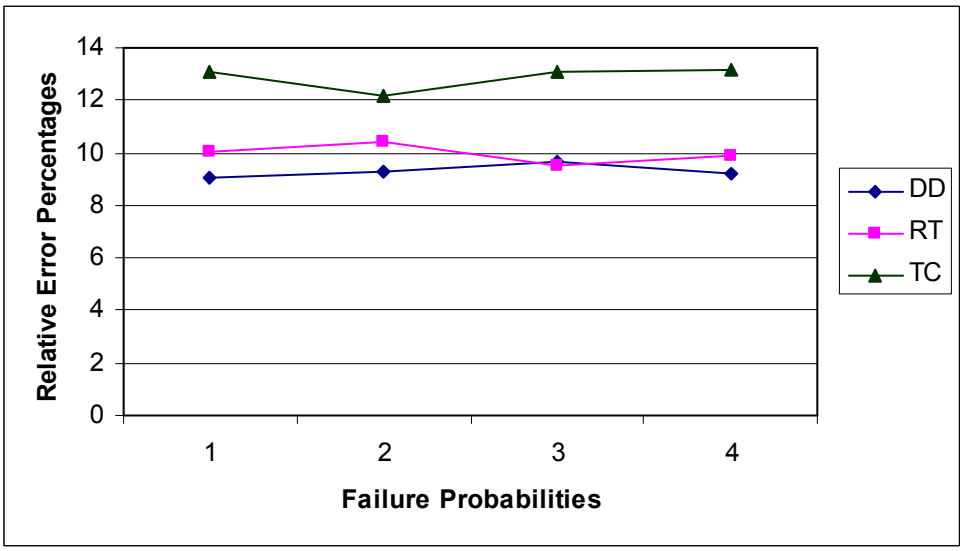


Figure 4-17 Variation of relative error percentages across different order of failure probabilities for a functional size of 75 FP

The error across functional sizes is progressively greater which is also expected because of the increase in the number of defects across functional sizes. Therefore the number of defects that contribute to the error (which is 20% of the total number of defects for the results provided above) also increases leading to an error bloat. The increase of error is sub-linear in nature.

Table 4-8 gives the variation of Requirements Traceability errors across different requirements traceability efficiencies. The upper bound of failure probability is 10E-1. The percentages of errors remained the same when the upper bound of failure probability was varied between 10E-2 and 10E-6. The results are the same for Defect Density errors when the inspection efficiency is varied accordingly. This is because the error functions for both the error models are the same (eq(4-1) and eq(4-6)).

Functional Size RT/Inspection Efficiency		75	150	300	600	1200	2400
60%	Mean	0.13	1.38	1.67	2.00	2.62	3.0
	Std. Dev	0.04	1.09	1.15	1.76	2.70	2.23
40%	Mean	0.43	1.47	1.75	2.43	2.94	3.43
	Std. Dev	0.14	1.98	1.23	1.24	1.65	2.12
20%	Mean	0.87	1.8	2.83	3.25	3.87	4.67
	Std. Dev	0.15	1.09	1.17	1.7	2.54	2.21
10%	Mean	1.19	2.12	3.4	3.9	4.5	5.74
	Std. Dev	0.19	1.68	2.80	2.56	2.17	1.64

Table 4-8 Mean and Standard Deviation for Requirements Traceability/DefectDensity error forms for varying functional sizes and requirements traceability/inspection efficiencies with upper bound of fault exposure probability of 10E-1 (Multiply each value by 10-1)**

Test Coverage error percentage follows a similar graph as that of the Defect Density and the Requirements Traceability errors. However, the error percentage is higher than that of the Defect Density and Requirements Traceability errors. This is

also expected since the number of defects detected through testing alone is less than the number of defects detected through inspection for the Defect Density errors. Moreover, for a Requirements Traceability defect removal efficiency of about 60%, Test Coverage and Requirements Traceability errors were similar. Figure 4-18 shows the variation of relative error percentages of Defect Density, Requirements Traceability and Test Coverage RePSs across different sizes. As can be seen, Test Coverage error follows a similar graph as that of Defect Density but the error percentage is higher for any specific size. Also, it shows that the relative error percentages for Defect Density, Requirements Traceability and Test Coverage errors increase across functional sizes.

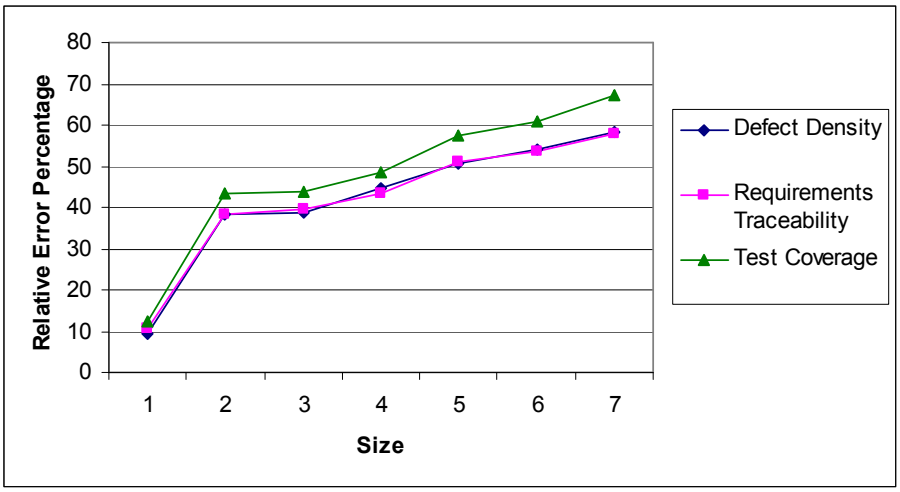


Figure 4-18 Relative error percentages for varying functional sizes with upper bound of fault exposure probability of 10E-2

When the testing efficiency for Test Coverage error is varied, the relative error percentage also varies. Figure 4-19 shows the variation of the Test Coverage relative error for different testing efficiencies. The X-axis represents the different sizes taken into consideration in increasing order.

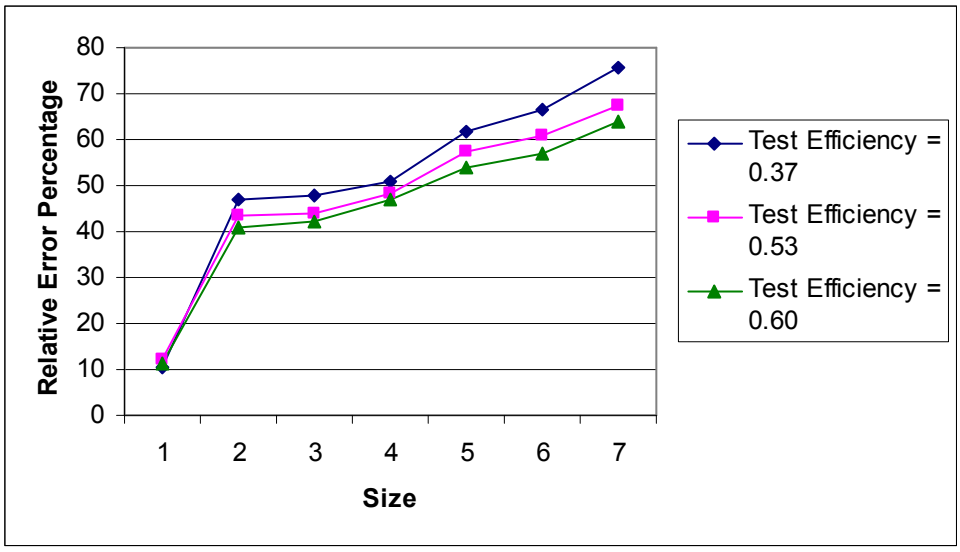


Figure 4-19 Test Coverage relative error percentages for varying functional sizes and varying testing efficiency with upper bound of fault exposure probability of 10E-2

When the percent of bug fix errors was changed from 5% to 8%, the results for test coverage errors did not change significantly. However, when the defect repair probability was varied, the errors changed significantly. The figure below shows the change in relative error percentages when the repair probability is changed. The test efficiency is 0.53. From Figures 4-20 and 4-21 it can be seen that the errors are similar for the applications with smaller sizes. The error difference across functional sizes gets progressively greater because of the rapid increase in the number of defects across functional sizes.

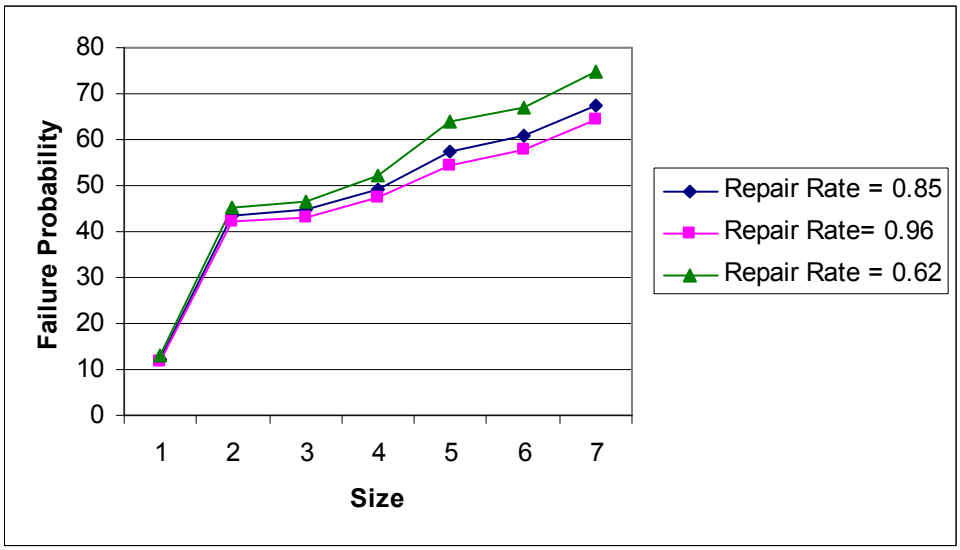


Figure 4-20 Test Coverage relative error percentages for varying functional sizes and varying repair probabilitys with upper bound of fault exposure probability of 10E-2

Also in the results provided above (Table 4-3 – Table 4-8) the standard deviation of k^* is $0.01k$. When the standard deviation increases, the error also increases but is not very significant. Figure 4-21 and 4-22 provides the relative error percentages of an application with failure probability of the order 10E-1 and functional sizes 75 and 10,000 function points respectively. The values 1, 2, 3, 4, 5 across the X-axis refer to the Defect Density, Requirements Traceability, Test Coverage, Bugs per Line of Code and Function Point error models respectively. There are noticeable differences in the Defect Density, Requirements Traceability and Test Coverage relative error percentages in case of the application with a functional size of 10,000 function points. This is again because of the increase in number of defects that contribute to the error.

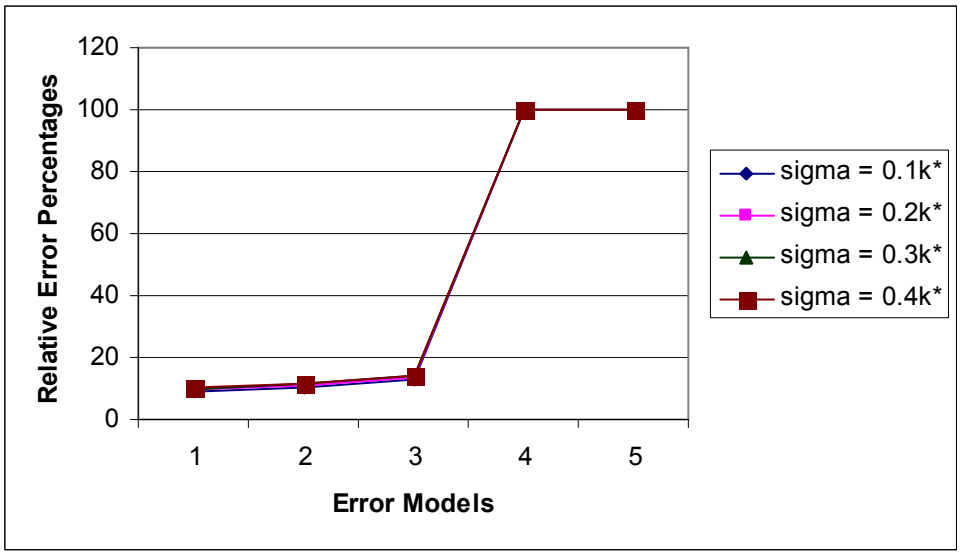


Figure 4-21 Relative error percentages for an application with failure probability of the order 10E-1and functional size of 75 FP

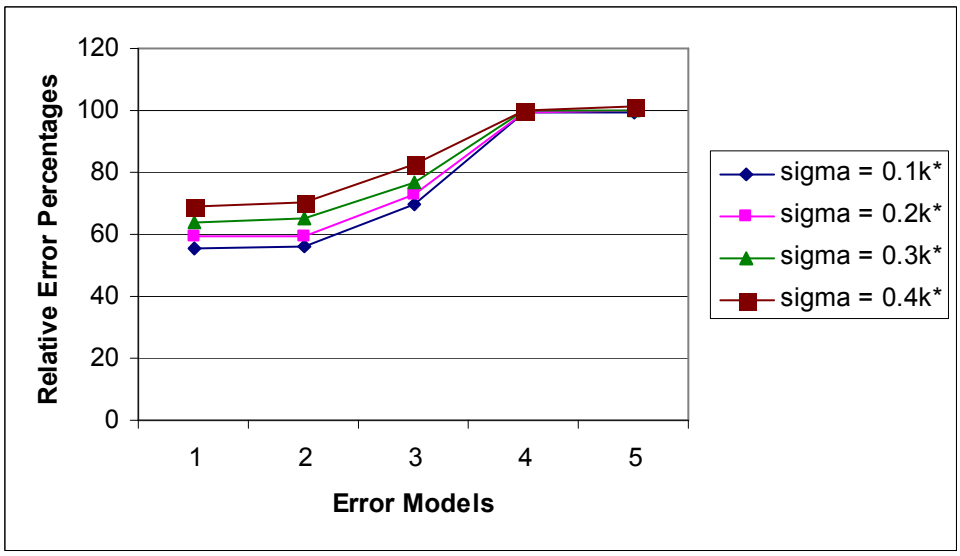


Figure 4-22 Relative error percentages for an application with failure probability of the order 10E-1 and functional size of 10,000 FP

However it is a very different situation for the Function Point and the Bugs per Line of Code errors. First, the error percentages are much higher than that of

Defect Density, Requirements Traceability and Test Coverage. The relative error percentages are almost 100% across applications with different functional sizes for failure probabilities of the order 10E-1 and 10E-2. This is because the probabilities of failures predicted by the two models are insignificant in these cases due to the prevalence of Musa's fault exposure ratio K (Section 3.1.4, Section3.1.5) whose value is of the order of 10E-7. Therefore the predicted probabilities are of the order of 10E-4 to 10E-6 (eq.(4-7) and eq.(4-9)). Please note that the $\frac{\tau}{T_L}$ ratio for the results provided above is considered to be unity. When the real failure probability is of the order of 10E-3, the prediction errors are lesser in case of applications of larger sizes. The reason is the same i.e. the predicted failure probabilities become more significant in these cases (eq.(4-7) and eq.(4-9)). In the cases where real failure probability is of the range 10E-4-10E-5, it can be seen that the prediction error is very little or negative. When the real failure probability is of the order of 10E-6 or lower, we get only negative errors and the relative error percentages is very high. Figure 4-23 shows this variation for different order of failure probabilities. All these observations follow the same rationale. The negative values stem from the fact that the predicted probabilities are higher than the real probabilities. These observations not only tell us that the prediction accuracies of the Function points and the Bugs per Line of Code RePSs are weaker, but also speak volumes about the irrelevance of Musa's fault exposure ratio K in most of the cases. They show

that it provides good estimations of failure probabilities for very specific application sizes and when the real failure probabilities are of a specific order.

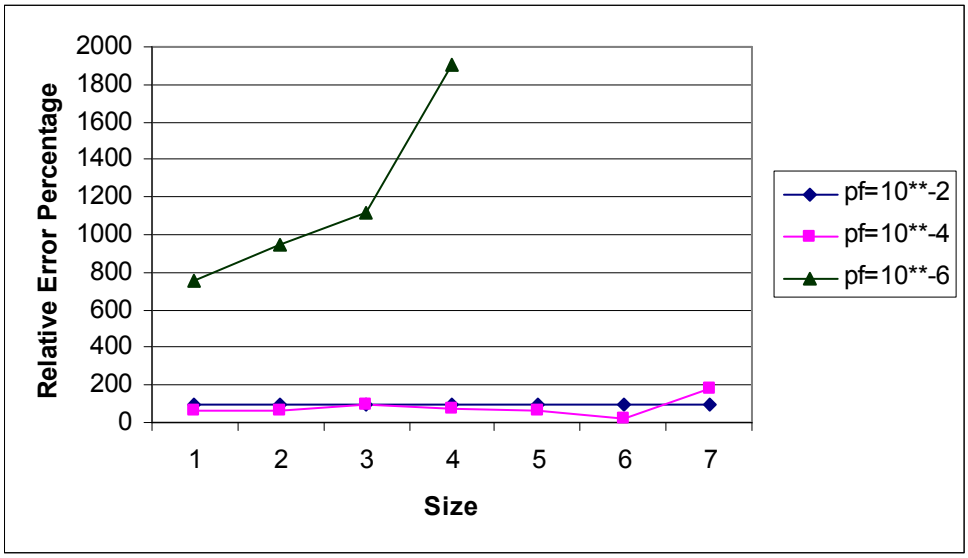


Figure 4-23 Relative error percentages for Function Point Error for applications of different order of failure probabilities

It is also observed that when the $\frac{\tau}{T_L}$ ratio was varied, the results were the same but for a different set of sizes and order of failure probabilities.

The results above (Table 4-3 through Table 4-8) have taken 200 lines of code per module into account for Bugs per Line of Code error values. When the number of lines of code per module was increased to 300 and 400, the predicted failure probabilities were lower (eq (4-9)) and hence the error results were slightly worse for higher orders of failure probabilities and were slightly better for lower orders of failure probabilities. Figure 4.24 and Figure 4.25 show the variation of the percentages of relative errors for 200 and 300 lines of code per module with 10E-1 and 10E-6 as upper bound of failure probabilities. The X-

axis refers to the different sizes in increasing order. However the results are not significantly different.

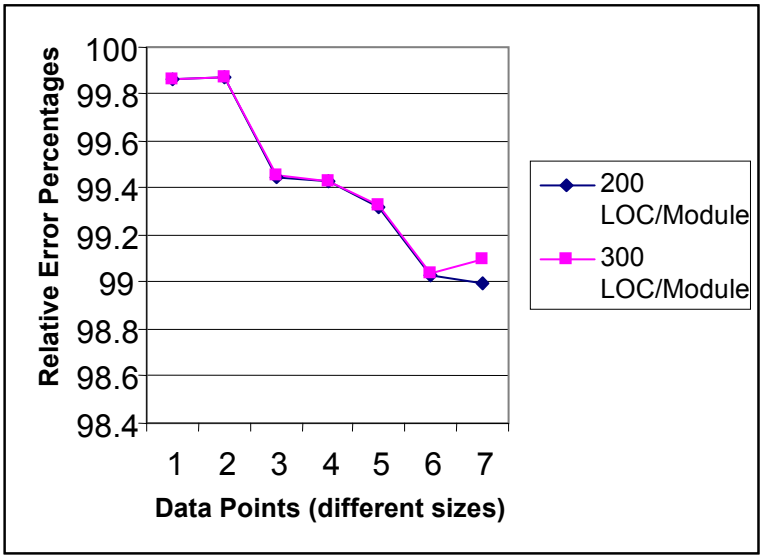


Figure 4-24 Relative error percentages for an upper bound of failure probability of 10E-1 and varying LOC per module

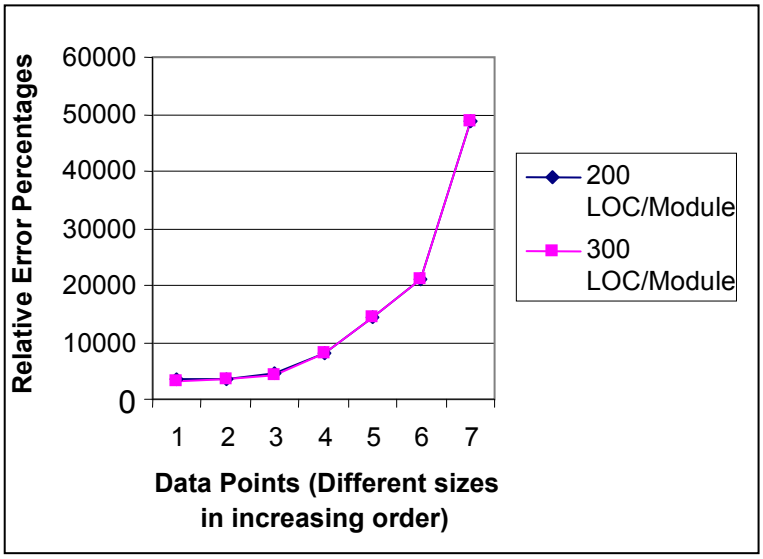


Figure 4-25 Relative error percentages for an upper bound of failure probability of 10E-6 and varying LOC per module

Moreover, the results provided in Table 4-3 through Table 4-8 have taken C++ as the coding language for Bugs per Line of Code error values. The median C++ backfiring coefficient as per Jones [Jones96] is 55. The errors varied only slightly when the coding language was changed to Visual Basic, FORTRAN or C. The backfiring coefficients for these languages are 32 (Visual Basic 3), 71 (FORTRAN 95) and 128 respectively. An increase in backfiring coefficient leads to an increase in number of modules (eq(4-9)) and hence an increase in the predicted failure probability. This leads to better estimates for higher order of failure probabilities and worse estimates for lower order failure probabilities (eq(4-9)).

4.4 Summary

In this chapter, the error models for the simulation were constructed. We believe that error models for simulation can be constructed in a similar fashion for any model with a thorough knowledge of the models. Simulation has been used to determine the nature of the errors in some engineering disciplines. [Jiao04] conducted a simulation to study the impacts of error structure on stock–recruitment (S-R) models. S-R models are fishery models that predict the amount of juvenile recruitment or production as a function of the parent stock. The authors observed through simulation that S-R models are less effective especially when there is a smaller data set. Simulation therefore provides a better alternative. [Kim98] conducted a simulation study to assess the impact and sources of errors in distributed decision support systems (DSS). DSS are computer technologies which

allow users to collect and analyze data in more sophisticated and complex ways and thus help in making a decision.

One would probably need to make some assumptions for the simulation, but they can be well augmented and justified either through previous work that exists or through validation/experimentation. Simulation is well worth the effort because of its flexibility and the range of applications it can simulate. Error values for any application can be generated through simulation.

Not only that, it helps understand the modeling approach better and identifies the fallacies of the models to a greater extent. It also is very effective from cost, time and effort perspective. Simulation may provide the only alternative in some cases, for example, an application with very low order of failure probability which may take hundreds of years [Butler93] of testing to estimate the failure probability. It is also a desirable and feasible alternative for very large applications.

Chapter 5 Experiment

An experiment was designed to validate the results obtained from the simulation on the prediction accuracies of the RePSs and the nature of their error models. This was important to establish the level of accuracy of each RePSs and confirm/reject/refine the error models associated with the RePSs. This was also consequential to validate the assumptions made in the simulation process. This chapter provides a detailed discussion of the experiment and the experimental results.

5.1 The Experiment Design

An in-vitro experiment was designed [Field03], [Hughes71] involving eight different applications to validate our findings. This section provides the objective, hypotheses, design, threats to validity, subjects and the execution of the experiment in details.

5.2 The Objectives of the Experiment

There were three main objectives of the experiment.

- to investigate the accuracy of the RePSs
- to determine the form of the "errors" ie whether they are additive or multiplicative in order to apply the information to the model uncertainty framework
- to validate the assumptions of the simulation

5.3 *Hypotheses of the Experiment*

This section gives the different sets of hypotheses that were used to verify the objectives.

5.3.1 First set of hypotheses

This section provides the hypotheses to statistically evaluate the prediction accuracies of the five RePSs. For this purpose, we defined a term 'relative error' (p_e) for each RePS which is equal to

$$p_e(\mathrm{Re}PS) = \frac{|\, p_s - p_s^*(\mathrm{Re}PS)\,|}{1 - p_s} \qquad (5\text{-}1)$$

Where

p_e (RePS) The relative error for a particular RePS

p_s The probability of success per demand obtained from reliability testing.

p_s^*(RePS) The probability of success per demand predicted by the particular RePS

This definition implies that the lower the value of p_e, the better the prediction. Statistical tests were then conducted on the relative error of each of the five RePSs.

As a rule of thumb, the relative error should ideally be less than or equal to one. This rule of thumb derives from the fact that a regulator will at least want a reliability estimate of the correct order of magnitude. However, we wanted to

further understand the limitations of each RePS and thus experimented on a broader set of hypotheses rather than confining ourselves to just one particular value. Therefore three sets of hypotheses were formulated. The first set of general hypothesis is that the relative error for each of the RePSs is less than equal to one; the second is that it is less than 0.50 and the third is that it is less than 0.30.

Table 5-1 gives the null (H_0) and alternate (H_A) hypotheses [Field03], [Hughes71] that the relative errors are less than unity.

RePS	Null Hypotheses	Alternate Hypotheses
Bugs per Line of Code	$H_{0BLOC} : \boldsymbol{p_e}_{BLOC} <= 1$	$H_{ABLOC} : \boldsymbol{p_e}_{BLOC} > 1$
Defect Density	$H_{0DD} : \boldsymbol{p_e}_{DD} <= 1$	$H_{ADD} : \boldsymbol{p_e}_{DD} > 1$
Function Point	$H_{0FP} : \boldsymbol{p_e}_{FP} <= 1$	$H_{AFP} : \boldsymbol{p_e}_{FP} > 1$
Requirements Traceability	$H_{0RT} : \boldsymbol{p_e}_{RT} <= 1$	$H_{ART} : \boldsymbol{p_e}_{RT} > 1$
Test Coverage	$H_{0TC} : \boldsymbol{p_e}_{TC} <= 1$	$H_{ATC} : \boldsymbol{p_e}_{TC} > 1$

Table 5-1 Null and alternate hypothesis that the relative errors is less than unity

Where $p_{e\,\boldsymbol{BLOC}}$ = relative error in the Bugs per Lines of code RePS model

$p_{e\,\boldsymbol{DD}}$ = relative error in the Defect Density RePS model

$p_{e\,\boldsymbol{FP}}$ = relative error in the Function Point RePS model

$p_{e\,\boldsymbol{TC}}$ = relative error in the Requirements Traceability RePS model

$p_{e\,\boldsymbol{TC}}$ = relative error in the Test Coverage RePS model

Similarly the null and alternate hypothesis that the relative error is less than 0.5 is given by Table 5-2.

RePS	Null Hypotheses	Alternate Hypotheses
Bugs per Line of Code	$H_{0BLOC} : p_{e\ BLOC} <= 0.5$	$H_{ABLOC} : p_{e\ BLOC} > 0.5$
Defect Density	$H_{0DD} : p_{e\ DD} <= 0.5$	$H_{ADD} : p_{e\ DD} > 0.5$
Function Point	$H_{0FP} : p_{e\ FP} <= 0.5$	$H_{AFP} : p_{e\ FP} > 0.5$
Requirements Traceability	$H_{0RT} : p_{e\ RT} <= 0.5$	$H_{ART} : p_{e\ RT} > 0.5$
Test Coverage	$H_{0TC} : p_{e\ TC} <= 0.5$	$H_{ATC} : p_{e\ TC} > 0.5$

Table 5-2 Null and alternate hypothesis that the relative error is less than 0.5

Table 5.3 gives the null and alternate hypothesis that the relative error is less than 0.3.

RePS	Null Hypotheses	Alternate Hypotheses
Bugs per Line of Code	$H_{0BLOC} : p_{e\ BLOC} <= 0.3$	$H_{ABLOC} : p_{e\ BLOC} > 0.3$
Defect Density	$H_{0DD} : p_{e\ DD} <= 0.3$	$H_{ADD} : p_{e\ DD} > 0.3$
Function Point	$H_{0FP} : p_{e\ FP} <= 0.3$	$H_{AFP} : p_{e\ FP} > 0.3$
Requirements Traceability	$H_{0RT} : p_{e\ RT} <= 0.3$	$H_{ART} : p_{e\ RT} > 0.3$
Test Coverage	$H_{0TC} : p_{e\ TC} <= 0.3$	$H_{ATC} : p_{e\ TC} > 0.3$

Table 5-3 Null and alternate hypothesis that the relative error is less than 0.3

5.3.2 Second set of hypotheses

The second objective of the experiment was to determine the form of the "errors" i.e. whether they are additive or multiplicative. The simulation results were used to build the general hypothesis.

Therefore the null (H_0) and alternate (H_A) hypotheses for the error models are given as:

Error Model	Null Hypotheses	Alternate Hypotheses
Bugs per Line of Code	H_{0BLOC} : e_{BLOC} is multiplicative	H_{ABLOC} : e_{BLOC} is not multiplicative
Defect Density	H_{0DD} : e_{DD} is additive	H_{ADD} : e_{DD} is not additive
Function Point	H_{0FP} : e_{FP} is multiplicative	H_{AFP} : e_{FP} is not multiplicative
Requirements Traceability	H_{0RT} : e_{RT} is additive	H_{ART} : e_{RT} is not additive
Test Coverage	H_{0TC} : e_{TC} is additive	H_{ATC} : e_{TC} is not additive

Table 5-4 Null and alternate hypotheses for the error models

Where $\boldsymbol{e_{BLOC}}$ = error in the Bugs per Lines of code RePS model

$\boldsymbol{e_{DD}}$ = error in the Defect Density RePS model

$\boldsymbol{e_{FP}}$ = error in the Function Point RePS model

$\boldsymbol{e_{RT}}$ = error in the Requirements Traceability RePS model

$\boldsymbol{e_{TC}}$ = error in the Test Coverage RePS model

In order to determine the error form, normality and lognormality tests were conducted on each of the error models.

5.3.3 Third set of hypotheses

The results obtained from simulation and those obtained from experiment were compared in order to validate the third objective. The general hypothesis is that they are equal.

Therefore the null (H_0) and alternate (H_A) hypothesis is given by

RePS	Null Hypotheses	Alternate Hypotheses
Bugs per Line of Code	$H_{0BLOC}: e_{simulation\ \mathrm{BLOC}} = e_{experiment\ BLOC}$	$H_{ABLOC}: e_{simulation\ \mathrm{BLOC}} \neq e_{experiment\ BLOC}$
Defect Density	$H_{0DD}: e_{simulation\ DD} = e_{experiment\ DD}$	$H_{ADD}: e_{simulation\ DD} \neq e_{experiment\ DD}$
Function Point	$H_{0FP}: e_{simulation\ FP} = e_{experiment\ FP}$	$H_{AFP}: e_{simulation\ FP} \neq e_{experiment\ FP}$
Requirements Traceability	$H_{0RT}: e_{simulation\ RT} = e_{experiment\ RT}$	$H_{ART}: e_{simulation\ RT} \neq e_{experiment\ RT}$
Test Coverage	$H_{0TC}: e_{simulation\ TC} = e_{experiment\ TC}$	$H_{ATC}: e_{simulation\ TC} \neq e_{experiment\ TC}$

Table 5-5 Null and alternate hypotheses that the simulation and the experimental errors are similar

Where $e_{simulation}\,\mathrm{Re}\,PS$ is the simulation error for a particular RePS and $e_{experiment}\,\mathrm{Re}\,PS$ is the experimental error for that RePS.

5.4 The Design

An in-vitro experiment was designed to achieve the objectives described in Section 5.2. The experiment was conducted in two parts. First was the development phase of different software applications. The second was the measurement phase where RePSs were constructed for the five models.

5.4.1 Design of the Development phase

This part was conducted in the class on Software Quality Assurance (SQA), a graduate course offered at the University of Maryland during Spring 2004. The SQA course consisted of 22 students, three of whom were off-campus students. The on-campus students were divided amongst five different groups, Group ATM, Group SRQS, Group WPU, Group SSP and Group LOCAT with six, five, four, two and two students each.

ATM, SRQS, WPU, SSP and LOCAT were the five different applications. ATM is the largest, SRQS and WPU were of medium sizes and SSP and LOCAT were of smaller sizes.

Each of these groups was divided into two subgroups. The experiment design for *each group* is shown below

	RR	SDesign	SDR	Coding	UT	CI	ST
Subgroup1	√		√	√	√	√	
Subgroup2	√	√	√			√	√

Table 5-6 Experiment design for development phase

where

RR: Software Requirements Review SDesign : Software Design

SDR : Software Design Review Coding : Software Coding

UT : Unit Test CI : Code Inspection

ST : System Test

The three off-campus students developed different versions of LOCAT individually. These three applications will henceforth be referred to as LOCAT-I, LOCAT-II and LOCAT-III.

The experiment design for off-campus students is

	RR	SDesign	SDR	Coding	UT	CI	ST
Student1	LOCAT-I LOCAT-II	LOCAT-II	LOCAT-I LOCAT-II	LOCAT-I	LOCAT-I	LOCAT-I LOCAT-II	LOCAT-III
Student2	LOCAT-II, LOCAT-III	LOCAT-III	LOCAT-II, LOCAT-III	LOCAT-II	LOCAT-II	LOCAT-II, LOCAT-III	LOCAT-I
Student3	LOCAT-III, LOCAT-I	LOCAT-I	LOCAT-III, LOCAT-I	LOCAT-III	LOCAT-III	LOCAT-III, LOCAT-I	LOCAT-II

Table 5-7 Experiment design for the off-campus students for the development phase

5.4.2 Design of the Measurement phase

The second part of this experiment, the measurement phase, was carried out in the (Software Reliability Engineering) SRE lab. Seven students were involved in this experiment. RePSs were constructed from five different measures: Requirements Traceability, Defect Density, Function Point, Test Coverage and Bugs per Line of Code for each of the eight applications. Each RePS had different numbers of support measures (ranging from one to seven). In total, more than 150 tasks were performed. By task we mean estimating the support measures, modeling the systems as Finite State Machine (FSM) models and computing the final

reliabilities. The design for the measurement phase is provided as an appendix (Appendix B).

The work was designed to avoid any bias. For example, support measures for the construction of Defect Density RePS and support measures for the construction of Requirements Traceability RePSs were measured by different subjects due to similarity in their structure. The similarity stems from the fact that both RePSs are defect oriented and the defects considered in the construction of RePS from Defect Density are a superset of the defects considered in the construction of RePS from Requirements Traceability. This is because the defects considered in the construction of RePS from Defect Density are found through inspection of requirements specifications, design specifications and the code where as the defects considered in the construction of RePS from Requirements Traceability are found by tracing the requirements to design and code. Moreover the construction of both the RePSs uses the EFSM technique to propagate the defects.

5.4.3 Threats to Validity

The experiment design minimizes the effects of threats to validity [Campbell63]. This section discusses threats to validity during the development phase and threats to validity during the measurement phase.

5.4.3.1 Threats to Validity during the Development Phase

Threats due to *selection* of respondents which can introduce disparity in development groups are eliminated through randomization. The students were

monitored through log sheets on their daily performances. This gave us a chance to observe any effects due to *History* and *Maturation*. [Campbell63]

One of the most prominent threats to external validity is caused by the use of students as subjects. However, the students were graduate students of computer engineering, computer sciences and electrical engineering disciplines, and many of them had part-time or full time jobs in software companies. (see Section 5.5.1.1). Most of them were familiar with software engineering concepts and had a lot of experience in software development. Moreover they were trained in course of the class on all issues pertaining to the experiment. Another threat to external validity is that of *representativeness* of the applications i.e. if they are representative of the real world scenario. Although this is an impossible task to achieve, the judicious choice of applications tries to alleviate this problem. The applications chosen are of varying sizes and come from different application domains that include database applications, real time applications, word processors etc. The applications were developed in different languages like C, C++, and VB.

5.4.3.2 Threats to Validity during the Measurement Phase

Here also, threats due to *selection* of respondents which can introduce disparity in measurement groups are eliminated through randomization. As before, the students were monitored through log sheets on their daily performances which gave us a chance to observe any effects due to *History* and *Maturation.* The effect of *instrumentation* is minimized because the measurements were performed according to unique guidelines and tools. As mentioned before, one of the most prominent threats to validity is caused by the use of students as subjects. For the

measurement phase all the students were from the SRE lab and were familiar with the measurement process and with the use of Testmaster [Testmaster99] and Winrunner [Winrunner01] which were the two main tools used. They were also given specific instructions and manuals etc on the measures and the measurement process.

The threats due to *repeatability* in the measures have also been taken care of. Even though there were a large number of common support measures [Smidts00, Li04], in each of the five RePSs, they were measured separately for repeatability concerns. Any discrepancies in the measures were analyzed, the measurement process was made more robust and unambiguous and the measurements were re-done if necessary.

5.5 *Experiment Execution*

This section discusses the execution process in details including the pre-experiment preparation and the final execution of the experiment.

5.5.1 Pre-experiment preparation for Development phase

In this section the steps taken as a pre-experiment preparation for the actual execution of the experiment are provided.

5.5.1.1 Subjects

The subjects for the development phase were students of a graduate level course on SQA at the University of Maryland. These students had comprehensive experience in the software development and testing field. Eight of these students were PhD students. There were three Computer Science graduate students, three

computer engineering graduate students, one electrical engineering graduate student, three systems engineering students and eight reliability engineering students. The following table provides information on the background of the students.

Profile	Number of Students
Currently working full time in software companies	4
Currently working part-time in software companies	2
Full time jobs in software industry in the past	4
Currently conducting research in the software field	18

Table 5-8 Subjects' Experience Profile

The students were not notified about the experiment to ensure that they would not be influenced by the knowledge of the experiment. The experiment was presented as a class project mandatory for the course, ensuring the necessary motivation. Preventive steps were taken to ensure that the students had no un-wanted communications during the course. Though it was an experiment per se, it was in line with the course and was conducted as a course project.

5.5.1.2 Applications

Applications from various domains, sizes and coding languages were chosen to ensure diversity and representativeness.

ATM [Ghose04C] or the Automated Teller Machine is the largest application of around 2500 SLOC. It was developed in Visual C++. The software performs the following activities:

Verification of customer identification, selection of services, deposition of cash or check, withdrawal of cash, transfer of funds between the customers'

accounts and inquiry of customers' account balances.

SRQS and WPU are the two medium sized applications of sizes of around 1500 SLOC each. SRQS [Ghose04D] or the Student Registry Query System is a database application and is designed for students to create and manage their accounts online. Registry DB is a database that maintains student SSN, student login ID, student password, course information, and registration information. SRQS generates SQL queries for retrieval of data from Registry DB. The result of the query is returned to the user interface.

SRQS performs the following activities: Create a student account, manage it by adding or dropping a course, view a particular course's schedule, booklist, waitlist etc and edit Registry DB by authorized personnel. This application was developed in Visual Basic.

WPU [Ghose04E] or the Word Processing Unit was designed to perform word processing functions such as adding text, deleting text, checking for errors, counting words and characters etc. The application performs these functions based on user inputs. User inputs are accepted either from direct keyboard entry or from ASCII source Input file. The output is written to a file. WPU was developed in C.

SSP [Ghose04B] or Small Search Program is a database application with around 600 SLOC. It has a database called Search PUBS created in Microsoft Access with information about authors and their publications. The database has three tables named *authors*, *titles* and *titleauthor*. The application asks the user for search options and generates SQL queries on the basis of the search options. SSP was developed in Visual Basic.

LOCAT [Ghose04A] is a real time simple projectile tracking system. It calculates the projectiles coordinates at any point of time. It was developed in Visual Basic and consists of around 500 SLOC.

LOCAT-I [Ghose04F], LOCAT-II [Ghose04G] and LOCAT-III [Ghose04H] are different versions of LOCAT and have different functions like finding range, angle, and velocity of the projectiles. LOCAT-I was developed in FORTRAN, LOCAT-II was developed in Visual Basic and LOCAT-III was developed in C++.

5.5.1.3 Groups

The students were given a questionnaire on the first day of the class. The questionnaire was mainly a background check of the students including their experience in the software field, the types of application/computer languages they have worked with, and the research that they were involved with. It also consisted of some coding and testing questions. Three groups were then formed: those proficient in C/C++, those proficient in VB and those conversant with testing. (Please note that C, C++ and VB were the only languages that were specified in the requirements specifications of the applications). Based on this information, the subjects were randomly assigned to a team/sub-group in a way so that each team was evenly balanced in terms of proficiency.

5.5.1.4 Execution

The experiment was run for a span of sixteen weeks. Each class, each week was 2 hrs and 40 min long. Students were trained before each round of assignments. Apart from the theory presentations, the sessions consisted of in class assignments. Questions were encouraged during the class but no interactions were allowed among

students outside the class. All questions to the instructor, outside the class were through e-mails or through help sessions. Events in the lecture and the help sessions were recorded and so were the questions through e-mails. The students were also given log-sheets and were demonstrated how to use them.

The subjects were provided with Software Requirements Specifications (SRS) of the applications and the experiment commenced with the review of these SRSs. They were formatted as per IEEE specification standards [IEEE84]. Every document, deliverable, review etc was formatted according to IEEE standards.

5.5.2 Pre-experiment preparation for the Measurement phase

In this section, the steps taken as a pre-experiment preparation for the actual execution of the measurement phase are provided.

5.5.2.1 Subjects

The "Measurement phase" is the phase of the experiment where RePS's were constructed from the five different software measures: Requirements Traceability, Defect Density, Function Point, Test Coverage and Bugs per Line of Code for each of the above eight applications. The subjects were seven graduate students of the Software Reliability Engineering Lab at the University of Maryland. Five of these were PhD candidates, one was a Post Doctoral student and one was a Masters student. All the students were conversant with different part/parts of the measurement process and were ideally suited for constructing RePSs.

5.5.2.2 Execution

As mentioned before more than 150 tasks were performed to construct the RePSs and compute the estimated failure probability. Detailed procedure to carry out the tasks is provided in [Smidts00]. These tasks were divided amongst the seven subjects. Care was taken to avoid any biases that were suspected to be present.

The subjects were first given a questionnaire to appraise their knowledge on the measurements. They were then given a lecture on the whole measurement process. This part of the experiment lasted for 8 weeks/two months. The design is provided in Appendix B.

The rules and regulations for the measurement phase were the same as those for the development phase. Questions were encouraged but no interactions were allowed among students. The students were also given log-sheets and were demonstrated how to use them.

The subjects were provided with requirements specifications, design specifications, code, and test plan of the applications as required, to perform their task. Moreover they were given manuals which had specific guidelines to carry out their task.

5.6 *Experiment Results*

This section provides the results of the experiment along with a statistical analysis of the results.

5.6.1 Statistical Analysis of the results to accept/reject the first set of hypotheses

The table below gives the final RePS values for the five different models and the real reliability values for each of the applications. Theoretically, it is not possible to compute the real reliability of an application. *Therefore, by real reliability we mean the experimental reliability value that is obtained through extensive testing*.

The steps taken to estimate this value are:

1) Construction of an EFSM [Wang93, Li06] representing the user's requirements in detail and embedding user's operational profile information. Testmaster tool [Testmaster99] was used for this purpose. This model is also the oracle for the application as the modeler is the person who knows the correct user specifications.

2) Execution of the model to evaluate the impact of the defects. A large number of test cases are run through the application and the ratio of number of test cases failed over the total number of test cases run, gives the real failure probability. Test cases were executed using *Winrunner* [Winrunner01].

Through reliability testing, it is taken care that all the defects that were discovered during the process of RePS constructions are represented and taken into account. Therefore, it is equivalent to four different persons testing the application. The first person discovers defects through inspection of the requirements, design and the code, the second person discovers defects through requirements traceability and the third person discovers defects through system testing. The fourth person is

the one that models the oracle and does extensive testing using Winrunner. Therefore, the application is not only tested by four different persons, it is also tested from different perspectives focusing in different techniques of finding defects. Boundary value analysis was performed for all the applications. Also, during this testing, enough test cases were generated not only to ensure complete functional testing of the application but also to ensure an all- path testing from the perspective of user's requirements, i.e., all the paths of the EFSMs were tested. The following table provides the details of the reliability testing. The 95% confidence interval and the standard error of reliability are determined assuming a binomial process.

Applications	Number of Test Cases	Number of Failed Test Cases	Estimated Reliability	95% Confidence Interval for Reliability	Standard Error of Reliability
ATM	300	8	0.97	(0.95,0.99)	0.009
SRQS	198	100	0.49	(0.42,0.56)	0.035
SSP	96	51	0.43	(0.33,0.53)	0.050
WPU	200	91	0.54	(0.47,0.61)	0.035
LOCAT	92	7	0.076	(0.022,0.130)	0.027
LOCAT_I	100	24	0.76	(0.67,0.84)	0.042
LOCAT_II	22000	0	1	(1,1)	0.0
LOCAT_III	50	6	0.88	(0.78,0.97)	0.045

Table 5-9 Details of the reliability testing of the applications

Applications	BLOC	DD	FP	RT	TC	Real Reliability
ATM	0.999998	0.99805	1	0.9956	0.94995	.9733
SRQS	0.9992074	0.95346	0.99998	0.98548	0.7334	.4949
SSP	0.962761	0.69059	0.99993		0.68259	.46875
WPU	0.999938	0.627	1	0.58119		.545
LOCAT	0.9986872	0.4	0.99993	0.1	0.10955	.0761
LOCAT_I	0.9965977	1	0.99935	1	1	.76
LOCAT_II	0.9999982	1	1	1	1	1.00
LOCAT_III	0.9999931	0.89	0.99982	1	1	.88

Table 5-10 Real reliability and reliability values predicted by the different RePSs

Table 5-10 provides the real reliability and the reliability values as estimated by the five RePSs for all the applications. The columns BLOC, DD, FP, RT, and TC refer to the reliability values obtained from the Bugs per Lines of Code, Defect Density, Function Point, Requirements Traceability and Test Coverage RePS respectively. The Real Reliability value corresponds to the experimental value which is obtained after extensive testing.

The missing value in case of Requirements Traceability RePS for SSP is due to the fact that the SSP application could not be compiled in the measurer's machine due to platform/operating system issues and the missing value in case of Test Coverage RePS of WPU is because the measurer was not able to find a bug in the application. And this RePS is based on the assumption that the application has at least one bug. Moreover, no defects were found in LOCAT-II and the real reliability

was computed as one. Since, theoretically it is not possible to have a completely bug-free application, it was considered an outlier. Therefore, this application was not taken into consideration for the statistical analysis.

The errors were computed from the above table as the difference between real reliability and the predicted reliability.

Table 5-11 provides the error data and their mean and standard deviation for each application for each of the RePSs. Table 5-12 provides the relative error for each of the RePSs for each application. Relative error is defined as the ratio of the error to the real value of failure probability.

Applications	Bugs/LOC	DD	FP	RT	TC
ATM	0.026	0.024	0.02	0.022	-0.02
SRQS	0.50	0.45	0.50	0.49	0.23
SSP	0.49	0.22	0.53		0.21
WPU	0.45	0.082	0.45	0.03	
LOCAT	0.92	0.32	0.92	0.023	0.03
LOCAT_I	0.23	0.24	0.23	0.24	0.24
LOCAT_III	0.11	0.01	0.11	0.2	0.12
Mean	0.39	0.19	0.40	0.16	0.13
Standard Deviation	0.30	0.16	0.30	0.18	0.11

Table 5-11 Error Data and Their Mean and Standard Deviation

Applications	$p_{e\ BLOC}$	$p_{e\ DD}$	$p_{e\ FP}$	$p_{e\ RT}$	$p_{e\ TC}$
ATM	0.99	0.92	0.99	0.83	0.87
SRQS	0.99	0.90	0.99	0.97	0.47
SSP	0.92	0.41	0.99		0.402
WPU	0.99	0.18	1	0.079	
LOCAT	0.99	0.35	0.99	0.025	0.036
LOCAT_I	0.98	1	0.99	1	1
LOCAT_III	0.99	0.083	0.99	1	1

Table 5-12 Relative errors for each of the five models for each application

Statistical significance tests are conducted to test the first set of hypotheses with α= 0.05. The SAS tool is used for all the tests. Normality tests are performed to assess the normality of the data. If the data is normal, a one sample t-test allows us to test whether a sample mean significantly differs from the hypothesized value. [Field 03].

If the data is not normal, non-parametric Sign tests are performed to infer on the null-hypothesis. The results show that only Defect Density and Test Coverage data follow a normal distribution.

The results are given in the tables below. The H_{0RePS}: $p_{e\ RePS} <= 1$, H_{0RePS} : $p_{e\ RePS} <= 0.5$, H_{0RePS} : $p_{e\ RePS} <= 0.30$ denote first set of hypotheses. Estimate refers to the mean value of the relative error and DF denotes the number of degrees of freedom. t-value is the t-test statistic and M-value is the Sign test statistic.

Error Model	Estimate	DF	t-value H_{0RePS} : $p_{e\ RePS} <= 1$	Pr > |t| H_{0RePS} : $p_{e\ RePS} <= 1$	t-value H_{0RePS} : $p_{e\ RePS} <= 0.5$	Pr > |t| H_{0RePS} : $p_{e\ RePS} <= 0.5$	t-value H_{0RePS} : $p_{e\ RePS} <= 0.30$	Pr > |t| H_{0RePS} : $p_{e\ RePS} <= 0.30$
$p_{e\ DD}$	0.55	6	-3.08	0.98	0.36	0.36	1.73	0.066
$p_{e\ TC}$	0.63	5	-2.31	0.96	0.82	0.22	2.07	0.046

Table 5-13 Statistics of the t-tests on the first set of hypotheses

Error Model	Estimate	DF	M-value H_{0RePS} : $p_{e\ RePS} <= 1$	Pr > |M| H_{0RePS} : $p_{e\ RePS} <= 1$	M-value H_{0RePS} : $p_{e\ RePS} <= 0.5$	Pr > |M| H_{0RePS} : $p_{e\ RePS} <= 0.5$	M-value H_{0RePS} : $p_{e\ RePS} <= 0.30$	Pr > |M| H_{0RePS} : $p_{e\ RePS} <= 0.30$
$p_{e\ BLOC}$	0.98	6	-3.5	0.99	3.5	0.0078	3.5	0.0078
$p_{e\ FP}$	0.99	6	-3.0	0.98	3.5	0.0078	3.5	0.0078
$p_{e\ RT}$	0.65	5	-2.0	0.93	1	0.34	1	0.34

Table 5-14 Statistics of the Sign tests on the first set of hypotheses

We can conclude from above that there isn't enough evidence to reject the null hypotheses (since p-value > 0.05) for all the five RePSs for the null hypotheses that $p_{e\ \boldsymbol{RePS}} <= 1$. For the null hypotheses that $p_{e\ \boldsymbol{RePS}} <= 0.5$, we reject the null hypotheses for $p_{e\ FP}$ and $p_{e\ BLOC}$ only and for the null hypotheses that $p_{e\ \boldsymbol{RePS}} <= 0.3$, we reject the null hypotheses for $p_{e\ BLOC}$, $p_{e\ FP}$, and $p_{e\ TC}$. From the above tests it is observed that the relative error for all the five models passed the basic criterion of being less than or equal to one which is encouraging. Among the five RePSs, the relative errors for only Defect Density and Requirements Traceability RePSs are less than equal to 0.30 and seem to have better predictive ability than the others. Even though Test Coverage RePS did not pass the test that its relative error is less

than 0.3, it produced much better results than Bugs per Lines of Code and Function Point RePSs.

5.6.2 Statistical Analysis of the results to accept/reject the second set of hypotheses

The second objective was to determine the form of the "errors" i.e. whether they are additive or multiplicative. Normality and log normality tests for the errors are conducted at α = .05. For the log normality tests, normality tests [Hughes71] on the natural logarithm of error values are carried out. The results are given below. The Stem-Leaf and Box Plots for each of the error models are also provided.

There isn't enough evidence to reject the null hypothesis that e_{BLOC} follows a multiplicative distribution and therefore we accept the null hypotheses. The statistics for the same are given below.

Tests for Normality

Test		--Statistic---	-----p Value------	
Shapiro-Wilk	W	0.88162	Pr < W	0.2337
Kolmogorov-Smirnov	D	0.252296	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.075755	Pr > W-Sq	0.2078
Anderson-Darling	A-Sq	0.44876	Pr > A-Sq	0.1979

```
Stem Leaf                     #          Boxplot
   9 2                        1             |
   8                                        |
   7                                        |
   6                                        |
   5 0                        1          +-----+
   4 59                       2          *-----*
   3                                     |  +  |
   2 4                        1          |     |
   1 2                        1          +-----+
   0 3                        1             |
     ----+----+----+----+
 Multiply Stem.Leaf by 10**-1
```

Figure 5-1 Statistics of the log-normality tests on e_{BLOC}

Similarly as seen below, there is enough evidence to assume that e_{DD} follows an additive distribution.

Tests for Normality

Test		--Statistic---		-----p Value------
Shapiro-Wilk	W	0.93455	Pr < W	0.5902
Kolmogorov-Smirnov	D	0.179819	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.03696	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.250781	Pr > A-Sq	>0.2500

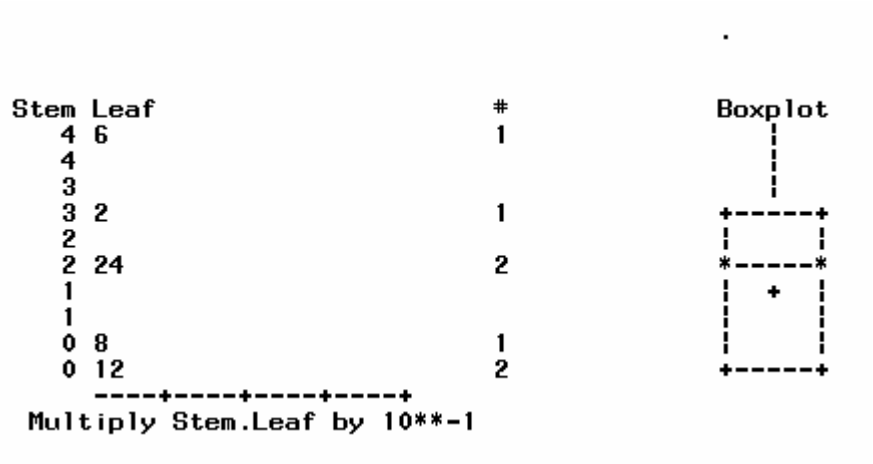


Figure 5-2 Statistics of the normality tests on e_{DD}

Also, we accept the hypotheses that e_{FP} follows a multiplicative distribution.

Tests for Normality

Test		--Statistic---		-----p Value------
Shapiro-Wilk	W	0.882868	Pr < W	0.2395
Kolmogorov-Smirnov	D	0.247676	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.073219	Pr > W-Sq	0.2238
Anderson-Darling	A-Sq	0.438175	Pr > A-Sq	0.2113

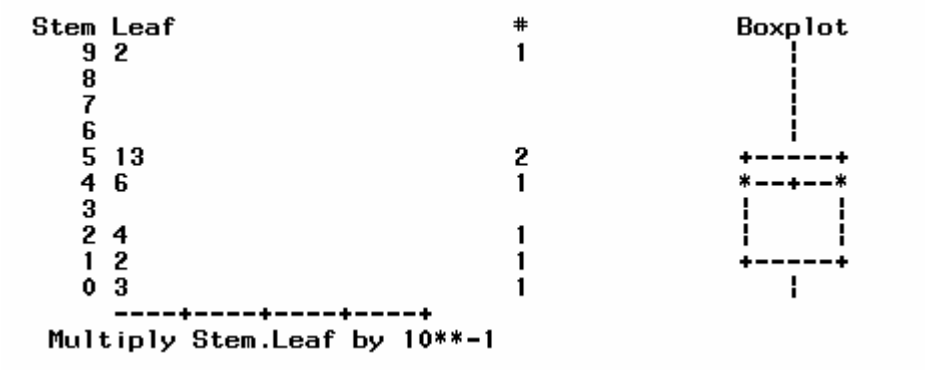


Figure 5-3 Statistics of the log-normality tests on e_{FP}

For e_{RT}, we accept the hypotheses that it follows an additive distribution. The statistics are given below.

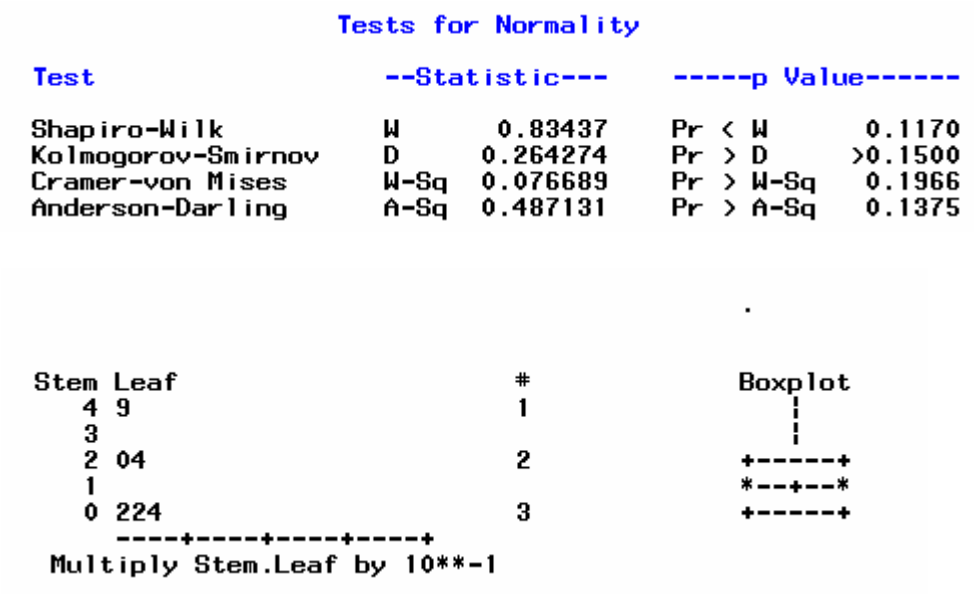


Figure 5-4 Statistics of the log-normality tests on e_{RT}

Similarly e_{TC} can be assumed to follow an additive distribution with the following statistics.

Tests for Normality

Test		--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.870389	Pr < W		0.2278
Kolmogorov-Smirnov	D	0.252063	Pr > D		>0.1500
Cramer-von Mises	W-Sq	0.061774	Pr > W-Sq		>0.2500
Anderson-Darling	A-Sq	0.387081	Pr > A-Sq		>0.2500

Stem	Leaf	#	Boxplot
2	144	3	+-----+
1			*-----*
1	2	1	| + |
0			| |
0	3	1	+-----+
-0	2	1	|

Multiply Stem.Leaf by 10**-1

Figure 5-5 Statistics of the log-normality tests on e_{TC}

5.6.3 Statistical Analysis of the results to accept/reject the third set of hypotheses

The third objective was to compare the error results obtained from simulation and from experiment to check if they are similar. The error form has been observed to be the same for the simulation and the experiment (Section 4.6.2). However we wanted to simulate the experimental data set and compare the results. We believe that doing so would give a better comparison as both the data set will be similar. Therefore equal number of data points with the same characteristics as that of the experiment were generated. By same characteristics, we mean that the sizes of the applications, their order of failure probabilities etc were kept the same for the simulation as that of the experiment. The set of characteristics for each application is given in Table 6-1.

Table 5-14 provides the mean and standard deviation of the simulation results

.

RePS	BLOC	DD	FP	RT	TC
Mean	0.48	0.20	0.46	0.19	0.15
Std. Dev	0.36	0.14	0.38	0.18	0.14

Table 5-15 Mean and standard deviation of the simulation results

Statistical significance tests were conducted to test the third set of hypotheses with α= 0.05. An F-test [Snedecor89] was used to test if the standard deviations of the two populations (errors from the simulation and errors from the experiment) are equal. The results showed that they are equal. t-tests were conducted for the data set that followed normal distribution i.e. Defect Density, Requirements Traceability and Test Coverage error data. Wilcoxon two-sample non-parametric tests were conducted for non-normal data i.e. Bugs per line of code and Function Point error data. The results are given in the tables below. t-value is the t-test statistic, W-value is the Wilcoxon test statistic.

RePS	**t-value** $H_{0RePS}: e_{simulation}\,\mathrm{Re}\,PS = e_{experiment}\,\mathrm{Re}\,PS$	**Pr > |t|** $H_{0RePS}: e_{simulation}\,\mathrm{Re}\,PS = e_{experiment}\,\mathrm{Re}\,PS$
DD	-0.08	0.6
RT	-0.35	0.6
TC	-0.31	0.6

Table 5-16 Statistics of the t-tests on the third set of hypotheses

RePS	W-value H_{0RePS} : $e_{simulation\,\mathrm{Re}\,PS} = e_{\exp eriment\,\mathrm{Re}\,PS}$	Pr > |W| H_{0RePS} : $e_{simulation\,\mathrm{Re}\,PS} = e_{\exp eriment\,\mathrm{Re}\,PS}$
BLOC	-1.022	0.30
FP	-0.50	0.61

Table 5-17 Statistics of the Wilcoxon-tests on the third set of hypotheses

We can conclude from above that there isn't enough evidence to reject the null hypotheses (since p-value > 0.05) for all the five RePSs for the third set of hypotheses. Therefore, simulation and experimental errors for all the five RePSs not only follow the same distribution but also have similar values of errors which is encouraging. However, further research may be conducted to analyze the simulation results for applications with larger sizes and smaller order of failure probabilities.

5.7 Summary

In summary, it was seen that the Defect Density, Requirements Traceability, and Test Coverage performed better than the Bugs per Line of Code and Function Point RePSs. It was also observed that the Defect Density, Requirements Traceability, and Test Coverage error models follow an additive distribution and Bugs per Line of Code and Function Point error models follow a multiplicative distribution. Moreover results also showed that there wasn't any significant difference between the findings from the simulation and the findings from the experiment.

Chapter 6 Updating the Estimations Based on Error Forms

In this chapter, we illustrate the technique of the uncertainty quantification procedure. First a general Bayesian Framework [Droguett02] to update a model's estimate in case of additive and multiplicative error models is presented. This is a generalized version of the Uncertainty Factor approach. The framework is then applied to the software applications considered in this study as examples. The results obtained from the application of the framework are then presented.

6.1 *Additive Error Model*

If there are n experimental results x_1^e ,....., x_n^e and corresponding model estimates are x_1^* , ...,x_n^* and the form of the error is known, it can be used to construct the likelihood function.

The model estimate is considered as a random variable, X , which is the sum of the true but unknown value, x, and a random error term E: $X^* = x + E$. In terms of realizations (i =1, …,n) of the random variables X^*, we have $x_i^* = x_i^t + E_i$ where x_i^* and E_i are realizations of the random variables X^*, *and E respectively,* (i =1, …,n). x_i^t is the true value of quantity X at i . Therefore, each realization E_i of E represents the difference between the model's estimate x_i^* and the true value of quantity X at i, x_i^t.

Under the assumption of no experimental error, $x_i^e = x_i^t$.

Thus, evidence on model M might be given as $D = \{E_1,\ldots,E_n\}$ where E_i is the error factor of i.

Now $L(x^* \mid \underline{\theta}, x)$ is a parametric likelihood function where the set of parameters $\underline{\theta}$ is estimated from the performance data $D = \{E_1,\ldots,E_n\}$ via Bayes' theorem. A simple, flexible, and practical form for the likelihood function $L(x^* \mid \underline{\theta}, x)$ is a Normal distribution with mean obtained as $x+b$, *where b =average(E)* is the bias factor and standard deviation σ. The set of parameters is now given by $\underline{\theta} = \{b, \sigma\}$. The likelihood can now be written as

$$L(x^* \mid x, \underline{\theta}) = L(x^* \mid x, b, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x^* - (x+b)}{\sigma}\right)^2} \tag{6-1}$$

Now, the posterior distribution of the set of parameters $\underline{\theta} = \{b, \sigma\}$ is

$$\pi(b, \sigma \mid E_1,\ldots,E_n) = \frac{L(E_1,\ldots,E_n \mid b, \sigma)\pi_0(b, \sigma)}{\int_b \int_\sigma L(E_1,\ldots,E_n \mid b, \sigma)\pi_0(b, \sigma) db d\sigma} \tag{6-2}$$

where the likelihood function is constructed considering that each pair of experimental results and corresponding model estimates for each realization i are independent, that is, $\{E_1,\ldots,E_n\}$ are independent realizations of the random variable E. Therefore,

$$L(E_1,\ldots,E_n \mid b, \sigma) = \prod_{i=1}^{n} L(E_i \mid b, \sigma) \tag{6-3}$$

Using the additive error model ,

$$L(E_1,\ldots,E_n \mid b, \sigma) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-1/2\left(\frac{(E_i - b)}{\sigma}\right)^2} \tag{6-4}$$

Now substituting in (5-2)

$$\pi(b,\sigma \mid E_1,...,E_n) = \frac{1}{k_1}\prod_{i=1}^{n}\frac{1}{\sigma}e^{-1/2(\frac{E_i-b}{\sigma})^2}\pi_0(b,\sigma) \qquad (6\text{-}5)$$

where $k_1 = \int_b\int_\sigma\prod_{i=1}^{n}\frac{1}{\sigma}e^{-1/2(\frac{E_i-b}{\sigma})^2}\pi_0(b,\sigma)dbd\sigma$ is a normalizing constant and $\pi_0(b,\sigma)$ is the prior distribution on *b* and *σ*.

Therefore now the likelihood function $L(x^*|D,x)$ is given as

$$L(x^* \mid D,x) = \int_b\int_\sigma \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x^*-(x+b)}{\sigma})^2}\frac{1}{k_1}\prod_{i=1}^{n}\frac{1}{\sigma\sqrt{2\pi}}e^{-1/2(\frac{E_i-b)}{\sigma})^2}\pi_0(b,\sigma) \qquad (6\text{-}6)$$

6.2 *Multiplicative Error Model*

The development of the procedure using the multiplicative error model is analogous to the additive error model, with the modification that the model estimate *X* is now modeled as the product of the true but unknown value, *x*, and an error term *E*:

Therefore, for each realization of X^*, where $x_i^* = x_i^t E_i$ where x_i^* and E_i are realizations of the random variables X^*, and *E* respectively, (i =1, …,*n*) and x_i^t is the true value of quantity *X* at *i*. The multiplicative error term is now given as $E_i = x_i^*/x_i^t$. Under the restriction of no experimental error, we have that $E_i = x_i^*/x_i^e$. By taking logarithms, $lnX^* = lnX + lnE$ as in the previous case, the likelihood function is given as

$$L(x^* \mid x,\underline{\theta}) = L(x^* \mid x,b,\sigma) = \frac{1}{\sigma x^*\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{\ln x^*-(\ln x+\ln b)}{\sigma})^2} \qquad (6\text{-}7)$$

Now, if $\pi o(b, \sigma)$ is a prior distribution of b and σ, the posterior distribution of the set of parameters $\underline{\theta} = \{b, \sigma\}$ is

$$\pi(b,\sigma \mid E_1,...,E_n) = \frac{1}{k_2}\prod_{i=1}^{n}\frac{1}{\sigma}e^{-1/2(\frac{\ln E_i - \ln b}{\sigma})^2}\pi_0(b,\sigma) \qquad (6\text{-}8)$$

where k_2 is normalizing constant and the likelihood function is constructed considering that each pair of experimental results and corresponding model estimates for each realization i are independent, that is, $\{E_1,\ldots,E_n\}$ are independent realizations of the random variable E.

Therefore, the likelihood function becomes

$$L(x^* \mid D,x) = \int_b\int_\sigma \frac{1}{\sigma x^*\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{\ln x^* - (\ln x + \ln b)}{\sigma})^2}\ \frac{1}{k_2}\prod_{i=1}^{n}\frac{1}{\sigma\sqrt{2\pi}}e^{-1/2(\frac{\ln E_i - \ln b)}{\sigma})^2}\pi_0(b,\sigma) \quad (6\text{-}9)$$

6.3 *Examples*

In this section we illustrate the uncertainty assessment of the software reliability of the applications given the estimates by the five RePSs. In order to assess the uncertainty of the RePSs estimates, some assumptions are made as required by the model uncertainty framework [Droguett02]. Different data sets were generated using the same set of conditions/characteristics as that of the applications. Table 6-1 provides the set of characteristics of each of the applications.

A homogenous population assumption is made concerning the error data. Given that the data sets that were used to update the failure probability prediction comprised only of results from the same set of characteristics, this is a reasonable assumption.

Applications	Size (LOC)	Language	Real Failure Probability	Defect removal efficiency of Inspection and Reviews (%)	Requirements Traceability Efficiency (%)	Defect detection efficiency of testing (%)	Defect repair probability (%)	$\frac{\tau}{T_L}$ ratio
ATM	2500	VC++	0.0267	60	60	100	80	6.86
SRQS	1500	Visual Basic	0.5051	18	9	56	54	0.44
WPU	1500	C	0.53125	78	71	N/A	N/A	0.889
SSP	600	Visual Basic	0.455	44	33	55	75	11.2
LOCAT	500	Visual Basic	0.9239	14	86	83	40	37.9
LOCAT_I	200	FORTRAN	0.24	0	0	0	N/A	1.28
LOCAT_III	250	C++	0.120	100	0	0	N/A	1.37

Table 6-1 Characteristics of the applications

Now the posterior distribution of the of the true unknown, *x*, (in this case the failure probability) given the available evidence D, is given as

$$\pi(x \mid x^*, D) = \frac{L(x^* \mid D, x)\pi_0(x)}{\int_x L(x^* \mid D, x)\pi_0(x)dx} \qquad (6\text{-}10)$$

Based on the simulation and the experimental results it is assumed that the error *E* representing the divergence between an estimate and an experimental measurement is described by the multiplicative error model for Bugs per Line of Code and Function point RePSs and additive error model for Defect Density, Requirements Traceability and Test Coverage RePSs. As a result, the parametric likelihood function for Bugs per Line of Code and Function point RePSs are modeled as Lognormal distributions and Defect Density, Requirements Traceability and Test Coverage RePSs are modeled as Normal distributions with parameters set $\theta = \{b, \sigma\}$, where *b* and σ are defined as before. Furthermore, let us consider that the

analyst's prior belief is negligible compared to the evidence provided by the performance data set D and the new estimate x^*. Thus, a flat prior $\pi_o(x)$ is adopted. Now, the mean value of posterior failure probability is given by

$$\int_x x \times \pi(x \mid x^*, D) dx \qquad (6\text{-}11)$$

The sections below provide examples and illustrate the different ways of obtaining an updated estimate.

6.3.1 Updates based on evidence on the order of failure probability

In this section, the process of application of model uncertainty framework based on evidence on the order of the failure probability of the application is discussed. Usually obtaining evidence on the order of failure probability of an application is tough as that would need comprehensive testing of the application. This also loses the purpose of estimation of failure probability. However, there may be cases like having different versions of the same application where there may be some idea on the order of failure probability of the application. Updates can then be made on the estimation of failure probability. Now, say that we know that SRQS has the same order of failure probability as that of SSP, WPU and LOCAT-I. Therefore the estimates for SRQS can be updated based on the experimental results obtained for SSP, WPU and LOCAT-I. Table 6-2 provides the updated predictions of failure probability for SRQS. Figure 6-1 provides the plots of the updated failure probability predictions vs. the initial failure probability predictions for SRQS. The five data points along the X-axis (1, 2, 3, 4, 5), refer to the five RePSs: Bugs per line of Code, Defect Density, Function Point, Requirements Traceability and Test

Coverage respectively. The updated values are much closer to the experimental value.

RePSs	Updated Predictions
Bugs per Line of Code	0.37
Defect Density	0.34
Function Point	0.38
Requirements Traceability	0.35
Test Coverage	0.53

Table 6-2 Updated predictions of failure probabilities of SRQS

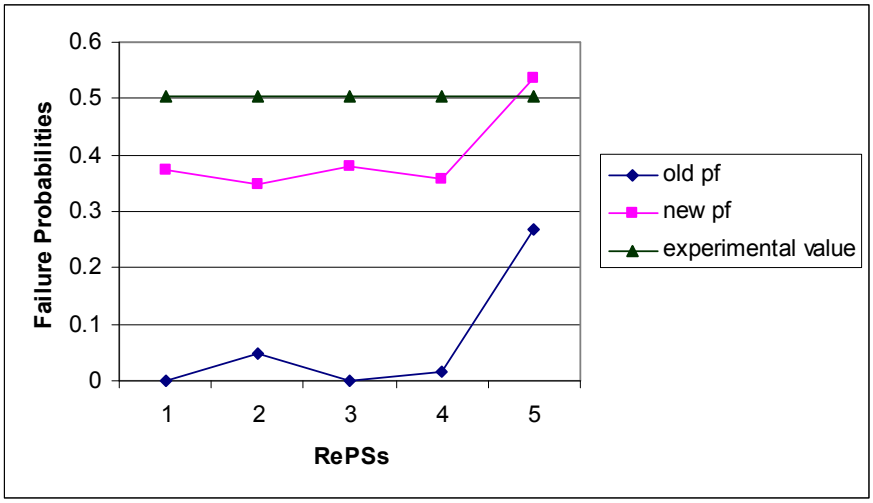


Figure 6-1 Plots of initial and updated values of failure probabilities of SRQS

6.3.2 Updates in the lack of any evidence

If there is no evidence at all of the order of the failure probability of an application, the values estimated by the RePSs may be considered as the upper bound of fault exposure probability. Figure 6-2 provides the plots of the updated failure probability predictions vs. the initial failure probability predictions for SRQS

considering the estimated values as the upper bound of fault exposure probability. The five data points along the X-axis (1, 2, 3, 4, 5), refer to the five RePSs as before: Bugs per line of Code, Defect Density, Function Point, Requirements Traceability and Test Coverage.

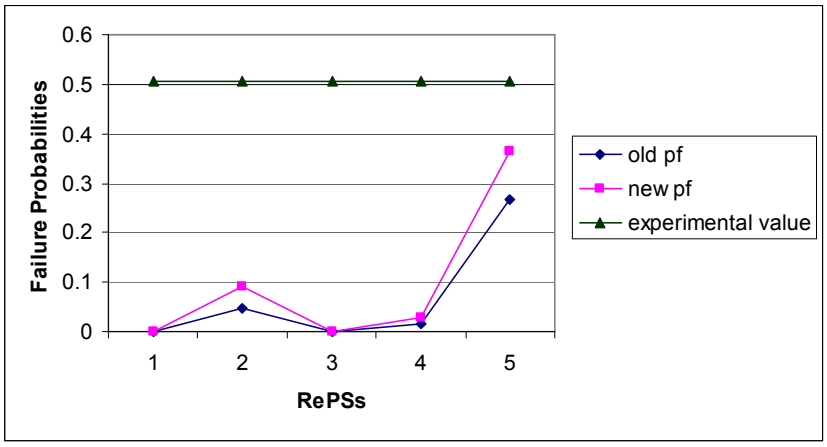


Figure 6-2 Plots of initial and updated values of failure probabilities

As can be seen, the updated estimates are better than the original estimates. However the degree to which the estimates get updated depends on how good the estimates already are. The table below provides the percentage of improvement in the estimates after they are updated. Percentage of improvement is defined as

$$\frac{updated\_estimate - initial\_estimate}{experimental\_value} * 100$$. As can be seen, the percentage of improvement is very small for the Bugs per Line of Code and Function Point errors. This is because the initial estimates are very low and close to zero. Therefore even a 100% improvement on the initial estimates is still close to zero.

	ATM	SRQS	WPU	SSP	LOCAT	LOCAT-I	LOCAT-III
DD	27	8	25	30	30	0	0
RT	28	2.6	33	N/A	11	0	0
TC	.06	19	N/A	28	.8	0	0
BLOC	1.7*10^-6	0.02	.07	.2	.3	.6	.34
FP	2*10^-5	2*10^-4	.08	.007	.035	.06	.66

Table 6-3 Percentages of improvement in the estimates based on no evidence of order of failure probability

6.3.3 Updates based on evidence on the accuracy of the models

It has been observed from the simulation and the experimental results that Defect Density, Requirements Traceability and Test Coverage RePS provide better estimates. Therefore in absence of any other evidence, the estimates provided by these RePSs can be averaged to obtain an idea regarding the real failure probability of the application and the estimates can be updated based on the averaged value. Figure 6-3 provides the plots of the updated failure probability predictions vs. the initial failure probability predictions for ATM. The five data points along the X-axis (1, 2, 3, 4, 5), refer to the five RePSs: Bugs per line of Code, Defect Density, Function Point, Requirements Traceability and Test Coverage respectively. The average value is the average of the Defect Density, Requirements Traceability and Test Coverage RePS estimates.

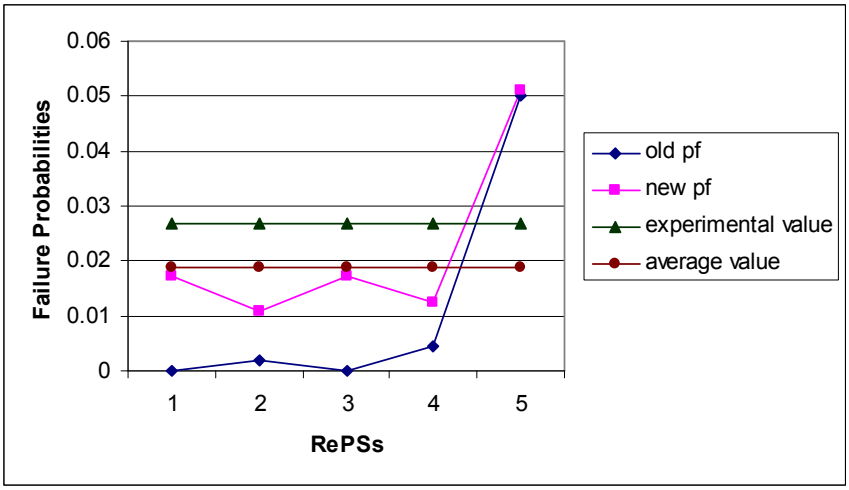


Figure 6-3 Plots of initial and updated values of failure probabilities of ATM

Here the failure probability for Test Coverage is overestimated initially. Since the defect detection efficiency of testing is 100% for ATM, the updated failure probability is similar to the initial failure probability. The failure probabilities for Bugs per Line of Code, Defect Density, Function Point and Requirements Traceability are updated as shown.

6.3.4 Updates based on evidence on the accuracy of the models and using an weighted average procedure

We now formalize the previous method of updating the estimates based on the average value of Defect Density, Requirements Traceability and Test Coverage RePS estimates. In this process, an weight is ascribed to each of the RePSs and an weighted average estimate is obtained. The procedure of making the updates is discussed below.

First, the average value of Defect Density, Requirements Traceability and Test Coverage RePS estimates, *AEst* (Average Estimate), is used to obtain the error

percentages of all the five RePSs. Based on the error percentages, a weight, W_{RePS}, is ascribed to each of the RePSs and then a weighted average estimate, *WAE*, is obtained for the application. Now this estimate can be used to make the final updates on the RePSs.

Formulating the weighted average procedure, the average estimate, of an application,

$$AEst = \frac{Initial\_Estimate_{DD} + Initial\_Estimate_{RT} + Initial\_Estimate_{TC}}{3} \quad (6\text{-}12)$$

Based on *AEst*, the average errors, *AErr* are determined for all the RePSs, for the specific application using simulation. Now, the average error ratio for an RePS, ER_{RePS}, is defined as the ratio of average error for that RePS over the average estimate AE, i.e. $ER_{\mathrm{Re}PS} = \frac{AErr(\mathrm{Re}PS)}{AEst}$ (6-13)

A new term, $Accuracy\_Index_{RePS}$, is coined to represent the accuracy of the RePS and is equal to ($1\text{-}ER_{RePS}$). If the accuracy index of a RePS is negative, it can be discarded. This is because as a rule of thumb, the average error should ideally be less than or equal to one. This rule of thumb derives from the fact that a regulator will at least want a reliability estimate of the order of magnitude [Li06].

The weight of the RePS, W_{RePS}, is equal to

$$W_{\mathrm{Re}PS} = \frac{Accuracy\_Index_{\mathrm{Re}PS}}{\sum_{\mathrm{Re}PS} Accuracy\_Index_{\mathrm{Re}PS}} \quad (6\text{-}14)$$

Therefore the weighted average estimate, *WAE,* is equal to,

$$WAE = \sum_{\mathrm{Re}PS} W_{\mathrm{Re}PS} * Initial\_Estimate_{\mathrm{Re}PS} \quad (6\text{-}15)$$

Simulation can then be run based on the weighted average estimate for each of the RePSs and the values can be updated accordingly. Figure 6-4 illustrates the weighted average update procedure.

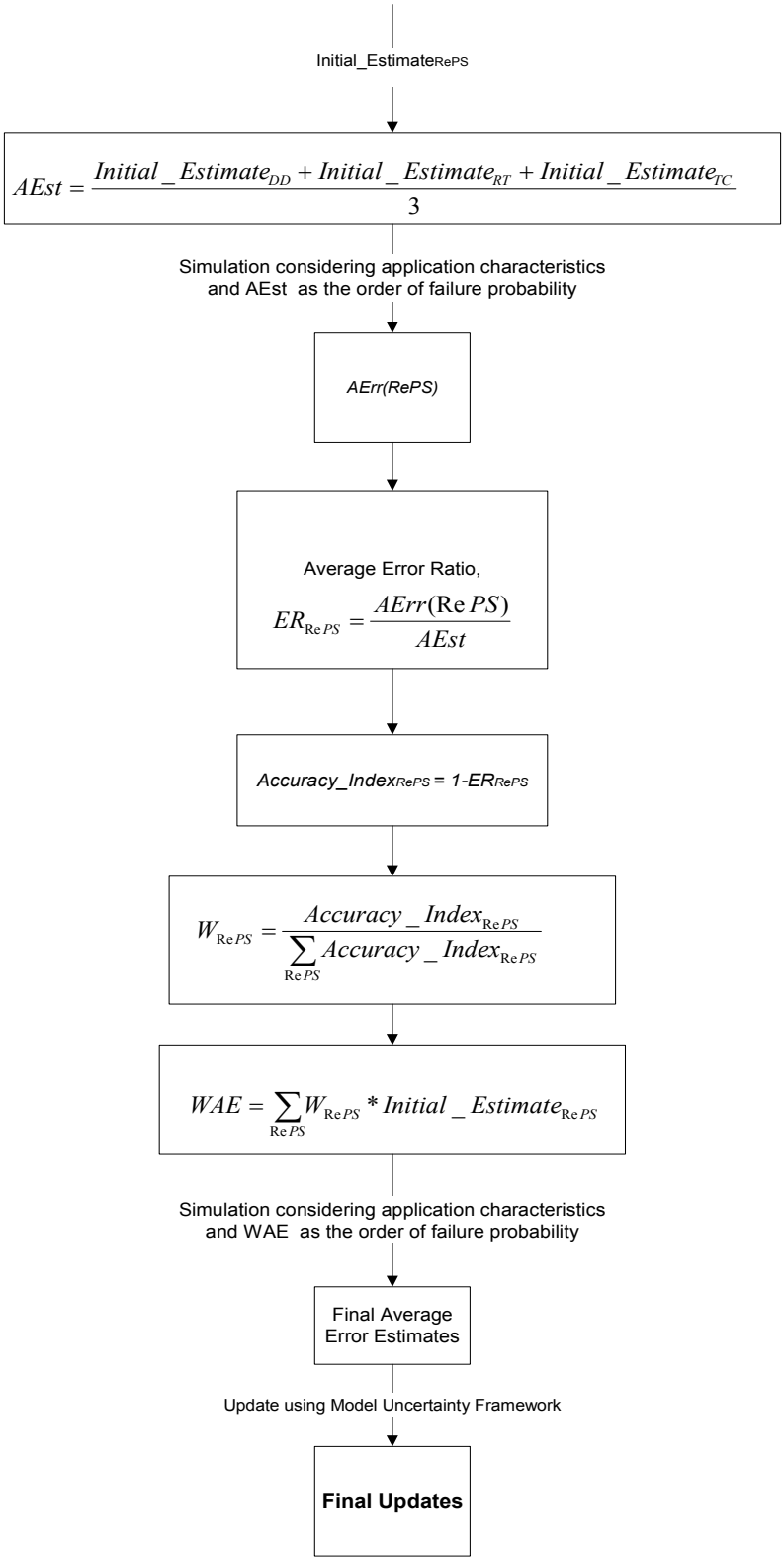


Figure 6-4 The weighted average update procedure

We now illustrate the weighted average procedure using examples. Let us consider the ATM application. The average estimate,

AEst = (0.00195+0.0044+0.05005)/3 = 0.0188 failure per demand.

Using the average estimate value as the order of failure probability and considering the characteristics of the application, (Appendix C), the average errors are obtained through simulation. The average errors are 0.018789, 0.00846, 0.018799, .00840 and 0.001 failures per demand for Bugs per Line of Code, Defect Density, Function Point, Requirements Traceability and Test Coverage RePS respectively. The error ratio for Bugs per Line of code, $ER_{BLOC,} = \frac{0.018789}{.0188} =$ 0.999415. Similarly the error ratios are 0.45, 0.999947, .446809 and .053191 for Defect Density, Function Point, Requirements Traceability and Test Coverage RePS respectively. The *Accuracy_Index* is now calculated as 0.000585, 0.55, 5.32E-05, 0.553191and 0.946809 for Bugs per Line of Code, Defect Density, Function Point, Requirements Traceability and Test Coverage RePS respectively.

Now the weights associated with each of the RePSs can be estimated. The weight for BLOC is equal to $\frac{0.000585}{0.000585 + .55 + 5.32E-05 + .553191 + .946809} = 0.000285.$

Similarly the weights are 0.268209, 2.59E-05, 0.269766, 0.461714 for Defect Density, Function Point, Requirements Traceability and Test Coverage RePS respectively.

Finally the weighted average estimate is calculated as 0.000285*2E-6 + 0.268209*.00195 + 2.59E-05*3E-6 + 0.269766*.0044 + 0.461714*0.05005 = 0.024819 failure per demand.

Based on the weighted average estimate, average errors are then obtained through simulation and the initial estimates are updated. The results of the updates are shown in Figure 6-5.

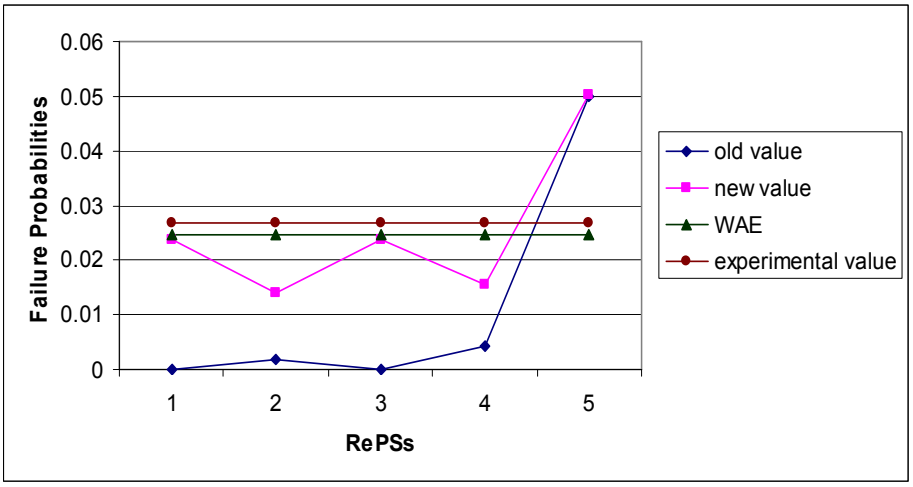


Figure 6-5 Plots of initial and updated values of failure probabilities of ATM taking WAE into account

The updated estimates are much closer to the experimental value. As can be seen from Figure 6-2 and 6-5, the updated estimates considering WAE are better than the updated estimates considering initial estimates by themselves.

The weighted average error procedure is now applied to the SRQS application. Table 6-2 provides the intermediate results for SRQS and Figure 6-5 shows the plots of initial and updated values. *AEst* for SRQS is 0.16383 and *WAE* is 0.19859.

RePS	Old Pf	ER_{RePS}	Accuracy_ Index	Weights	New Pf
Bugs per Line of Code	0.00079	0.99	2.13E-05	2.46E-05	0.19
Defect Density	0.0465	0.85	0.14	0.17	0.20
Funtion Point	2E-05	0.99	0.0001	0.00011	0.19
Requirements Traceability	0.0145	0.89	0.103	0.11	0.18
Test Coverage	0.266	0.38	0.61	0.70	0.34

Table 6-4 SRQS update results taking WAE into account

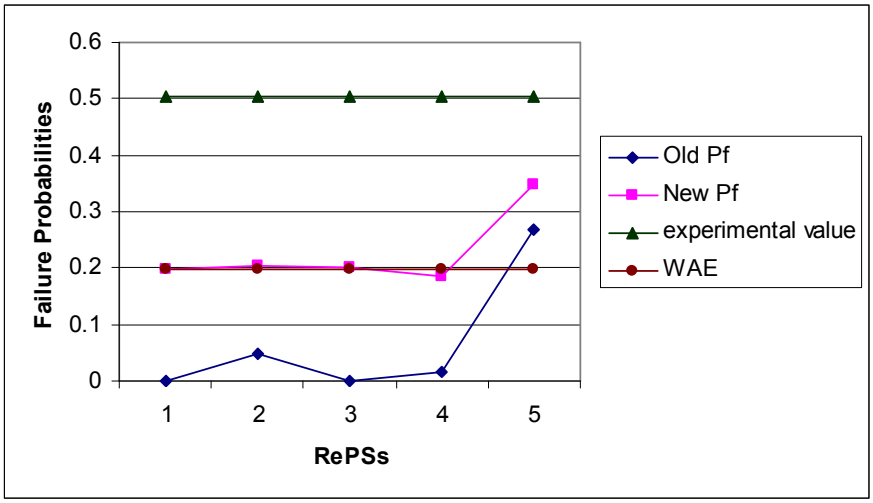


Figure 6-6 Plots of initial and updated values of failure probabilities of SRQS taking WAE into account

As seen from the results above, the updated values are much closer to the experimental value than the initial estimates. Figures 6-7 through 6-10 provide the initial and final estimates for SSP, WPU, LOCAT and LOCAT-III. The initial estimates for LOCAT-I for Defect Density, Requirements Traceability and Test Coverage are equal to zero. Therefore the initial average estimate (AEst) is also equal to zero and hence the updated estimates are also equal to zero. Table 6-5 provides the percentage of improvement in the estimates for all the applications

after they are updated. The four RePSs across X-axis for SSP (Figure 6-7) are Defect Density, Bugs per Line of Code, Function Point and Test Coverage respectively and the RePSs across X-axis for WPU are Defect Density, Bugs per Line of Code, Function Point and Requirements Traceability.

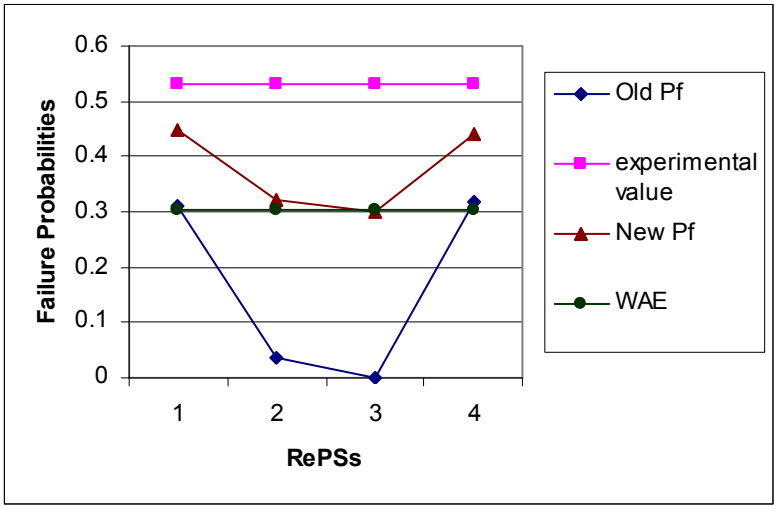


Figure 6-7 Initial and final estimates for SSP

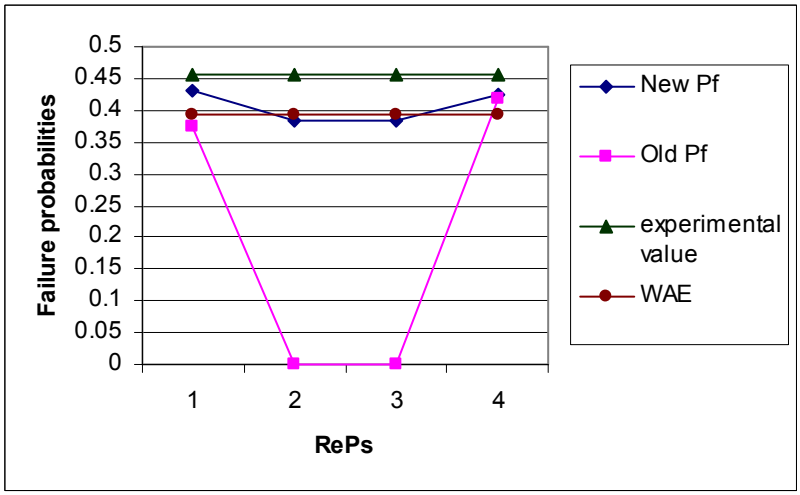


Figure 6-8 Initial and final estimates for WPU

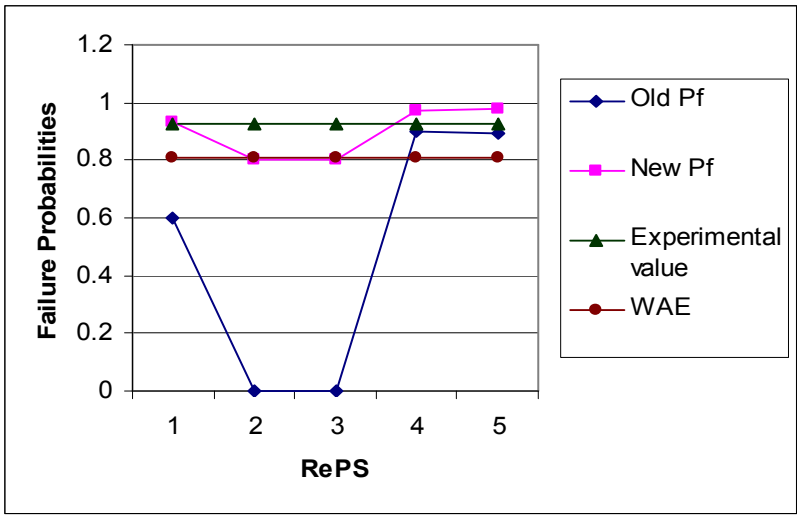


Figure 6-9 Initial and final estimates for LOCAT

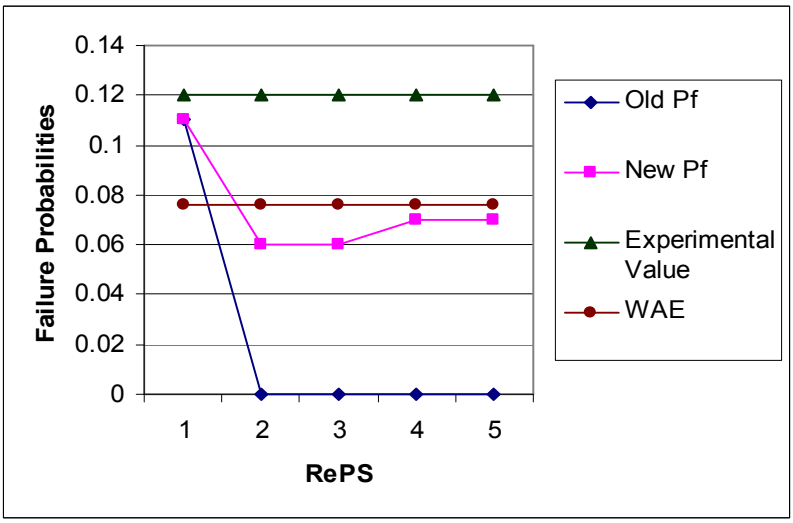


Figure 6-10 Initial and final estimates for LOCAT-III

RePSs	SRQS	ATM	WPU	SSP	LOCAT	LOCAT-I	LOCAT-III
BLOC	38.88	89.5	85	53	87	0	50
DD	31.35	44.76	12	26	35	0	0
FP	39.56	89.13	85	56	87	0	50
RT	33.61	41.27	0.26	N/A	8	0	58
TC	15.8	0.59	N/A	23	10	0	58

Table 6-5 The percentages of improvement in estimation

As mentioned before, the percentages of improvement for Locat-II are zero since *AEst* for Locat-II is equal to zero. The percentage of improvement for Bugs per Line of Code and Function Point ranges from 39% to 90%. The high percentage of improvement is because of the fact that the initial estimates were not good and were close to zero (Table 5-9). The percentage of improvement for Defect Density and Requirements Traceability varied from 0% to about 60%. The percentage of improvement are dependent on various factors such as how good the estimates already are and the specific set of characteristics of the application. For example, for Locat-III the initial estimate for Defect Density RePS is close to the experimental value (Table 5-9) and the defect removal efficiency is 100% (Table 6-1). Therefore the average error is zero and hence there is no improvement in the estimate. However in case of ATM, defect removal efficiency is 67% and therefore there is an error in the estimate and the estimate is improved by 44.67%. The percentage of improvement in test coverage RePSs also vary between 10% to 60%. The reasoning for the variation is the same as the Defect Density RePS variation.

Also, from Table 6-3 and Table 6-5, it can be seen that the percentage of improvement in estimation of RePSs are higher in case of the weighted average update procedure than in cases where just the initial estimates (Section 6.3.2) are used. Also, since evidence on the order of failure probability may not be available in most of the cases, the weighted average update procedure is recommended. The table below provides the initial average (AEst) and the weighted average (WAE) for each of the application.

	ATM	SRQS	SSP	WPU	LOCAT	LOCAT-I	LOCAT-III
AEst	0.018	0.11	0.31	0.39	0.79	0	0.036
WAE	0.026	0.19	0.30	0.39	0.81	0	0.076

Table 6-6 Initial average and weighted average estimates of failure probability

As far as recommending a single RePS is concerned, it depends on the characteristics of the application and/or the organization. It has already been observed from the experimental and simulation results that the Defect Density, Requirements Traceability and Test Coverage RePSs provide better estimations. According to Jones [Jones96], on an average the inspection procedure is 80% efficient in detecting faults. Therefore Defect Density RePS may be used to estimate the failure probability. The defects detected through Requirements Traceability RePS construction are a subset of the defects detected through Defect Density RePS (For reasoning, please refer to Section 5.4.2). However, detecting defects through Requirements Traceability is usually easier than detecting defects through inspection. Moreover, inspection procedures may not be followed strictly rendering the inspection less effective. In such cases Requirements Traceability RePS should be used. The historical data of the organization may be used to determine the efficiency of Defect Density and Requirements Traceability and the one with better efficiency should be used.

For the Test Coverage RePS, the testing efficiency as well as the test coverage ratio are instrumental in estimating the failure probability. It is seen from the simulation that with 53% testing efficiency and 70% test coverage, the relative error for Test Coverage RePS is similar to the relative error of Defect Density RePS at 60% inspection efficiency. (As illustrated in Figure 6-12, the third point in the Test Coverage plot corresponds to 70% test coverage and the corresponding equivalent

inspection efficiency is 60% (the third data point in the inspection efficiency plot)). Figures 6-11 through 6-13 provide the equivalent inspection efficiency for varying Test Efficiency and Test Coverage that provide similar relative errors in estimation of failure probability.

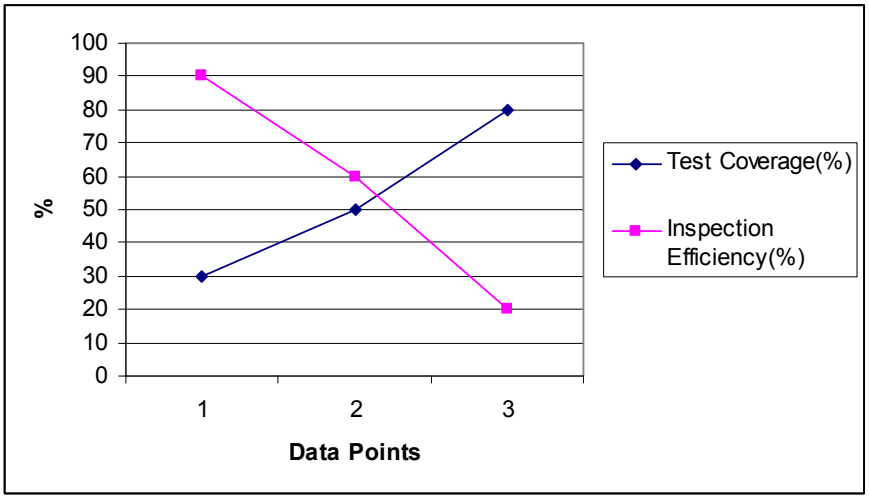


Figure 6-11 Inspection Efficiency for varying Test Coverage when the Test Efficiency is 30%

Figure 6-11 provides the inspection efficiency equivalent for varying test coverage when the test efficiency is 30%. From Figure 6-11, it can be seen that when the test efficiency is low (30%), and the test coverage increases, the error increases and therefore the equivalent inspection efficiency decreases. The error is the least when the test coverage is the same as the test efficiency, i.e, 30%, which means that when a 30% of test coverage detects the same percentage of faults, Test Coverage RePS provides good estimation. Please note that the estimated failure probability of the application will be high in this case since only 30% of the defects have been detected, but this estimation is closer to the real value and therefore the

error is small. In order for the Defect Density RePS to obtain such accuracy, the equivalent inspection efficiency is 90%.

Similarly, when test coverage is 80%, which also means that 80% of test coverage detected only 30% of the faults, the failure probability is under-estimated and the error in the failure probability estimation is very high. Hence the inspection efficiency equivalent that gives the same amount of error is 20%.

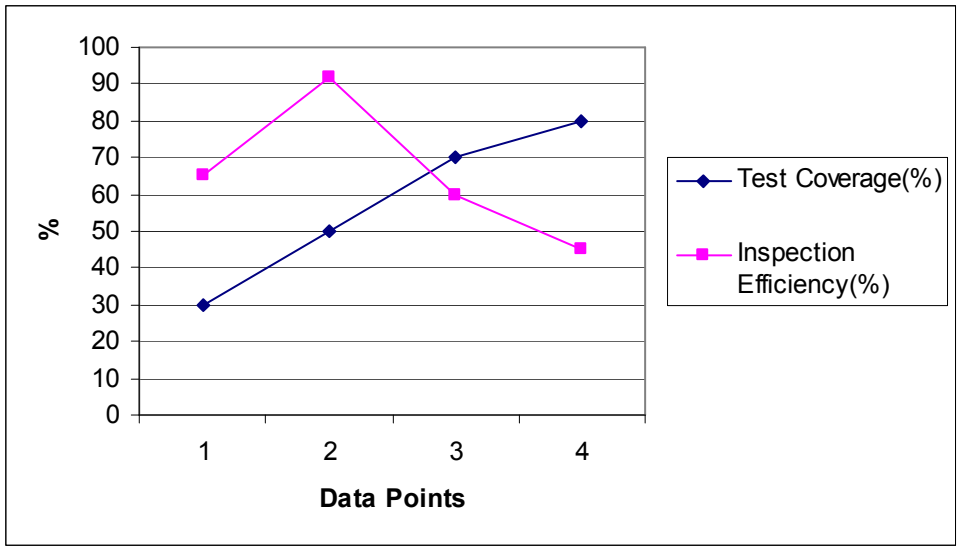


Figure 6-12 Inspection Efficiency for varying Test Coverage when the Test Efficiency is 50%

Figure 6-12 provides the inspection efficiency equivalent for varying test coverage when the test efficiency is 50%. As before, it can be seen that the error is the least when the test coverage is the same as the test efficiency, i.e, 50%, which means that a 50% of test coverage detects the same percentage of faults. Therefore, in order for the Defect Density RePS to obtain such accuracy, the equivalent inspection efficiency is about 90%.

Similarly, when test coverage is 80%, which also means that 80% of test coverage detected only 50% of the faults, the failure probability is under-estimated and the error in failure probability estimation is high. Hence the inspection efficiency equivalent that gives the same amount of error for the Defect Density RePS estimation is 45%. However when the test coverage is low (30%), which means that 30% of test coverage detected 50% of the faults, the failure probability is over-estimated and the inspection efficiency equivalent that gives the same amount of error in this case is 65%.

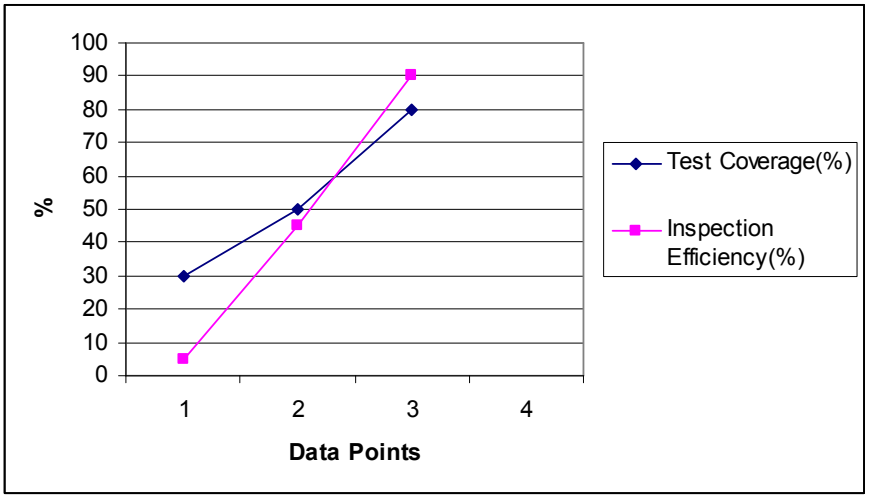


Figure 6-13 Inspection Efficiency for varying Test Coverage when the Test Efficiency is 80%

Figure 6-13 provides the inspection efficiency equivalent for varying test coverage when the test efficiency is 80%. As before, it can be seen that the error is the least when the test coverage is the same as the test efficiency. However, when the test coverage is low (30%), which means that 80% of the defects are detected by 30% of the test coverage, the Test Coverage RePS provides gross over-estimation of

the failure probability and the error is very high. An inspection efficiency that gives the same amount of error is 5%.

The results presented in the table above can be used as a guideline to determine which RePS to use. The historical data of the organization can be used to determine the testing efficiency and inspection efficiency and then the plots can be used to determine the RePS that provide a better estimate. Test Coverage can be determined using tools available commercially. For example, say that it is usually seen that a certain organization has typically a testing efficiency of 30%. The test coverage is measured as 60%. This means that the equivalent inspection efficiency that gives the same error is about 48% (Figure 6-11). Therefore if the organization typically has lesser inspection efficiency, it is better to use Test Coverage RePS. Else, Defect Density RePS should be used.

6.4 Summary:

As seen from the results, the updated failure probability values are much closer to the experimental value. The results also show that even in the absence of any evidence, the estimates can still be updated using simulation data. The updated estimates are better than the original estimates but the degree to which the estimates get updated depends on how good the estimates already are.

Moreover, updates can be made from the knowledge of the accuracy of the models even though there is no evidence like the order of failure probability of the application. Appropriate weights can be ascribed to each of the models and updates can be made accordingly.

We also would like to note that the approach of model improvement provided above can be generalized to other reliability models. The evidence on these models can be obtained either through experiment or through simulation. However we feel that the values obtained through simulation usually fit the homogenous assumption of the data set better. This is because the data is generated for a specific set of characteristics of the application. Moreover simulation saves time, effort and cost.

This chapter also provided guidelines to determine which RePS to use based on application/organization characteristics. The guidelines help choose the RePS with least relative error.

Chapter 7 Conclusion

We have achieved the objectives set out at the beginning of the dissertation. In this chapter we conclude by summarizing our achievements and the contributions made by this research. We also identify the limitations of our work. Avenues for future research are subsequently identified.

7.1 Contributions of this Research

- The research contributes to measurement-based software reliability modeling by statistically analyzing and validating five different software reliability prediction systems, RePSs. The RePSs are constructed from software engineering measures such as Defect Density, Bugs per Line of Code, Function Point, Requirements Traceability and Test Coverage. An experiment was designed to obtain data on the RePSs and statistically investigate their prediction performance.
- It generalizes the domain of application of model uncertainty approaches to the field of software reliability modeling. As illustrated in Chapter 6, a Bayesian approach of model uncertainty quantification has been used to update the estimates of failure probability. Therefore, the first steps to foray into the domain of model uncertainty have been taken in this research.
- It contributes to software reliability modeling intellect and the industry by providing more robust reliability estimates. As shown in Table 6-5, the updated estimates of failure probability were better than the initial estimates.

The percentage of improvement for Bugs per Line of Code and Function Point ranges from 39% to 90%. The percentage of improvement for Defect Density and Requirements Traceability varied from 0% to about 60%.

- We also provide an alternate approach of determining the error forms which is integral to the application of the model uncertainty framework: *simulation*. Simulation allows us to use a wide range of inputs. This acts as a catalyst for sensitivity analysis of the models and helps determining the error for different scenarios. The values assumed such as the defect removal efficiency etc can be fine-tuned as per the specific organization's software engineering practices. Traditionally the nature of the errors is determined experimentally. It includes the extensive and difficult task of designing the experiment in a way to counter threats to validity, and executing it. Also each experiment needs a minimum number of data points (the larger the number, the better) in order to statistically validate it. This is expensive not only from a cost perspective but also from a time perspective. Sometimes carrying out an experiment is just not feasible due to lack of resources. Simulation is an alternative which provides a solution to the above problems.
    - We also compared the results obtained from the simulation and from the experiment and found similarities between them. This corroborates our claim that simulation can be an alternative to determine the nature of errors.

- A tool was developed in the course of this research to simulate the error models. The tool can be used to estimate the errors on the failure probabilities predicted by the RePSs for applications of varying sizes, languages and domains. The error values can then be used to update and provide better predictions.

- The research also contributes to the understanding of some of the root causes of error in the software reliability model building processes leading to possible improvements of these. For example, it is almost customary to use Musa's K [Musa87, Musa98] in the estimation of software reliability. However in the course of simulation we realized the irrelevance of Musa's K in most of the cases. The simulation results showed that it can be used for a very specific order of failure probabilities and application sizes.

7.2 Limitations of this Research

Like everything else, this research has its limitations which are enumerated below.

- Assumptions in the simulation:
  - The assumption that the defects are independent of each other only excludes the case in which one defects masks others [Wu93]. This is a fair enough assumption as long as this situation does not happen frequently. However, this assumption may limit the research in some manner especially when there are many defects masking one another, and needs to be investigated. Moreover the real and the simulated

errors are assumed to be similar in this study. This assumption also needs to be investigated to provide better results.

  - Usage of defect removal efficiency values etc are only an average value and may not be reflective of the participating organization's own standards. Moreover they are dynamically changing due to the ever-changing software engineering paradigms and/or practices.
  - In Chapter 4, we use only certain values of defect removal efficiency, defect repair probability, backfiring coefficient and any other variable obtained from [Jones96]. This may lead to a certain amount of uncertainty on the error.

- We realize that the experimental data set is still very small. There are no data points on applications with very low failure probability or very large size. This may limit the inferences made especially on these kinds of applications. Moreover we are unable to compare the performance of the simulation at this level.

7.3 Future Work

Following are some possible areas of future research.

- Development of a larger data set: As mentioned before a large data set is essential and integral to validate any study. In the data set that is available, there are no applications with very low order of failure probabilities or very large sizes. Such applications should be included in the data set. This shall

allow for the scalability of the approach. Moreover it can be compared to the simulation data in a more appropriate manner.

- The assumptions in the simulation process such as independence of defects can be studied to better the estimate of errors in the simulation process. Also we assume that defects detected during software inspection are independent of their fault exposure probabilities as there are no evidences to believe otherwise. However this area may be explored to confirm such assumptions.
- We feel that this approach can be generalized to any software reliability model. Studies may be done to validate this hypothesis. From an organization's perspective this would allow for a better estimation of reliability while using the technique existing in the organization.
- Further work may also include correlation among models. A major issue in the assessment of model uncertainty is the possibility of dependence among the models considered in the weighted average error assessment. Possible reasons for dependence among models include [Droguett02]:
    - As they are representations of the same reality aspect, the models might share common theoretical principles,
    - Common implementation procedures, such as mathematical approximations,
    - They might have been conceptualized and implemented by individuals sharing the same basic training,
    - As a result of sharing similar modeling processes, they would share, to some degree, the available limited information

sources and have the tendency to be redundant with regards to some of their inputs.

The dependence of models in the model uncertainty framework should be studied and incorporated into the framework.

- Extension of error modeling to software systems with redundant components. In safety critical systems, a fault tolerance approach may be employed to obtain high reliability. The basic strategy of the software fault-tolerance approach is to design several versions of a program from the same specification and to employ a voter of some kind to protect the system from bugs. The voter can be an acceptance test (i.e., recovery blocks) or a comparator (i.e., N-version programming) [Butler93]. This system does not fail unless there is a co-incident error i.e. both the versions produce erroneous outputs in response to the same input [Butler93]. This research was solely aimed at software systems with single components and does not handle software systems with redundant systems. Further work may be done to extend error modeling to such systems and analyze the errors in the estimation of reliability of such systems.

Appendix A: Results of the simulation of error forms

This appendix provides the mean and standard deviation of the results obtained for different sets of simulation.

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	.099	1.14	1.30	1.77	2.39	2.96	4.56
	Std. Dev	.044	1.42	1.09	1.21	1.73	2.66	3.89
Requirements Traceability	Mean	.104	1.13	1.29	1.79	2.52	2.81	4.51
	Std. Dev	.046	1.10	1.13	1.23	1.58	2.64	3.78
Test Coverage	Mean	.11	1.42	1.67	2.09	2.69	3.27	5.46
	Std. Dev	.041	1.07	1.16	1.60	2.69	2.28	4.83
Function Point	Mean	1.15	2.02	3.70	3.97	4.36	5.60	9.87
	Std. Dev	1.99	1.68	2.80	2.56	2.17	1.64	2.55
Bugs per Line of Code	Mean	1.25	2.26	3.50	3.76	4.47	5.53	9.56
	Std. Dev	1.90	1.71	3.07	3.00	2.22	1.56	2.22

Table A-1 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-1 (Multiply each value by 10-1)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	0.12	1.24	1.56	1.80	2.16	2.57	5.01
	Std. Dev	0.12	1.31	1.60	1.74	2.04	2.80	2.16
Requirements Traceability	Mean	0.12	1.40	1.56	1.80	2.34	2.70	4.92
	Std. Dev	0.13	1.13	1.57	1.68	2.02	2.94	2.12
Test Coverage	Mean	0.14	1.55	1.77	2.04	2.46	2.83	5.14
	Std. Dev	0.14	1.54	1.57	1.88	2.18	2.98	2.19
Function Point	Mean	1.73	2.19	3.46	3.92	4.23	4.88	7.09
	Std. Dev	1.81	2.05	3.40	3.99	4.79	5.25	4.04
Bugs per Line of Code	Mean	1.55	2.09	3.20	3.33	3.57	3.84	5.12
	Std. Dev	1.60	1.94	3.28	3.58	4.24	5.12	2.66

Table A-2 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-3 (Multiply each value by 10-3)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	.080	1.01	1.38	1.65	2.43	3.01	4.66
	Std. Dev	0.034	1.15	1.34	1.34	1.35	2.39	2.67
Requirements Traceability	Mean	0.11	1.14	1.28	1.56	2.13	2.24	4.57
	Std. Dev	0.033	1.16	1.17	1.65	1.98	1.99	2.00
Test Coverage	Mean	0.14	1.48	1.69	2.00	2.54	3.17	5.82
	Std. Dev	0.054	1.12	1.16	1.87	2.35	2.98	2.83
Function Point	Mean	1.06	1.90	3.38	3.18	2.84	-1.24	-18.23
	Std. Dev	1.76	1.67	2.34	2.24	2.98	1.98	10.65
Bugs per Line of Code	Mean	.93	1.61	1.69	.65	-1.90	-6.48	-38.12
	Std. Dev	1.76	1.88	1.76	2.87	1.92	3.78	12.54

Table A-3 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-4 (Multiply each value by 10-4)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	.07	1.13	1.28	1.66	2.28	2.56	4.87
	Std. Dev	.034	1.12	1.09	1.56	1.89	2.97	3.09
Requirements Traceability	Mean	0.10	1.24	1.43	1.70	2.25	2.89	4.86
	Std. Dev	0.65	1.34	1.57	1.36	1.99	2.76	2.13
Test Coverage	Mean	0.12	1.37	1.70	1.99	2.77	3.34	5.97
	Std. Dev	.087	1.18	1.14	1.70	2.01	2.00	2.83
Function Point	Mean	.15	.0026	-0.54	-4.20	-22.04	-51.15	-278.02
	Std. Dev	1.88	1.65	2.98	2.65	2.18	1.94	22.66
Bugs per Line of Code	Mean	-1.35	-6.05	-12.50	-28.34	-60.54	-115.4	-476.76
	Std. Dev	1.89	1.67	3.23	3.07	8.33	12.99	29.45

Table A-4 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-5 (Multiply each value by 10-5)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	. 09	1.24	1.28	1.74	2.26	2.65	4.78
	Std. Dev	.045	1.32	.98	1.23	1.76	2.01	2.98
Requirements Traceability	Mean	0.10	1.24	1.41	1.80	2.30	2.79	4.78
	Std. Dev	0.04	1.31	1.11	1.36	1.98	2. 47	2.76
Test Coverage	Mean	0.12	1.34	1.69	1.99	2.45	3.02	5.69
	Std. Dev	0.06	1.23	1.13	1.76	1.86	2.03	2.87
Function Point	Mean	-9.35	-17.56	-39.45	-76.24	-255.3	-660.9	-2787
	Std. Dev	2.98	10.67	11.26	13.32	23.34	27.34	32.665
Bugs per Line of Code	Mean	-41.23	-81.36	-155.3	-315.3	-645.2	-1184	-4789
	Std. Dev	3.86	8.67	11.43	20.59	24.298	34.453	39.234

Table A-5 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-6 (Multiply each value by 10-6)**

Functional Size / Error		75	150	300	600	1200	2400	10000
Defect Density	Mean	. 067	1.14	1.38	1.67	2.28	2.56	4.87
	Std. Dev	.02	1.23	1.02	1.53	1.87	2.17	2.92
Requirements Traceability	Mean	0.10	1.32	1.54	1.69	2.35	2.78	4.65
	Std. Dev	.076	1.33	1.17	1.35	1.99	2.13	2.14
Test Coverage	Mean	0.13	1.49	1.78	2.00	2.76	3.40	5.97
	Std. Dev	0. 09	1.13	1.13	1.34	2.00	2.19	2.82
Function Point	Mean	-1.2E8	-1.2E8	-1.2E8	-1.2E8	-1.2E9	1.2E9	1.2E10
	Std. Dev	19813	15674	29845	26653	218567	219887	929998
Bugs per Line of Code	Mean	-4.27E8	-4.27E8	-4.28E8	-4.28E8	-4.28E9	-4.28E9	-4.3E10
	Std. Dev	21813	18675	31987	34554	329877	314657	999456

Table A-6 Mean and Standard Deviation for different error forms for varying functional sizes with upper bound of fault exposure probability of 10E-13 (Multiply all values by 10-13)**

The table below provides the percentage of the relative error for varying functional sizes with upper bound of fault exposure probability of 10E-1. Relative error is defined as the ratio of the absolute value of the error over the real value of failure probability.

Functional Size / Error	75	150	300	600	1200	2400	10000
Defect Density	9.09	39.31	37.34	43.54	51.76	53.43	57.42
Requirements Traceability	10.03	39.14	39.35	44.65.	51.65	52.97	58.87
Test Coverage	13.06	43.45	44.56	52.65	59.65	61.45	65.43
Function Point	99.98	99.91	99.86	99.68	99.46	99.34	99.12
Bugs per Line of Code	99.99	99.99	99.96	99.96	99.86	99.87	99.32

Table A-7 Relative error percentages for varying functional sizes with upper bound of fault exposure probability of 10E-1

Appendix B: Experiment Design for the Measurement Phase

ATM			Sub1	Sub2	Sub3	Sub4	Sub5	Sub6	Sub7
	B/LOC								
		S_i			1				
		T_L			1				
		τ			1				
		p_f			1				
	DD								
		DD Defects			1				
		FSM, p_f			1				
	FP								
		FP		1					
		N		1					
		T_L		1					
		τ		1					
		p_f		1					
	RT								
		RT Defects		1					
		FSM,p_f		1					
	TC								
		FP			1				
		FP_{Miss}			1				
		LOC_{Tested}	1						
		LOC_{IMPL}	1						
		T_L	1						
		τ	1						
		FSM,p_f	1						
SRQS									
	B/LOC								
		S_i				1			
		T_L				1			
		τ				1			
		p_f				1			
	DD								
		DD Defects				1			
		FSM, p_f				1			
	FP								
		FP						1	
		N						1	

		T_L						1	
		τ						1	
		p_f						1	
	RT								
		RT Defects						1	
		FSM,p_f						1	
	TC								
		FP					1		
		FP_{Miss}					1		
		LOC_{Tested}	1						
		LOC_{IMPL}	1						
		T_L	1						
		τ	1						
		FSM,p_f	1						
WPU									
	B/LOC								
		S_i	1						
		T_L	1						
		τ	1						
		p_f	1						
	DD								
		DD defects							1
		FSM, p_f							1
	FP	**FP**							1
		N							1
		T_L							1
		τ							1
		p_f							1
	RT								
		RT Defects				1			
		FSM,p_f				1			
	TC								
		FP		1					
		FP_{Miss}		1					
		LOC_{Tested}		1					
		LOC_{IMPL}		1					
		T_L		1					
		τ		1					
		FSM,p_f		1					
SSP									
	B/LOC								
		S_i				1			
		T_L				1			
		τ				1			

		p_f				1			
	DD								
		DD Defects				1			
		FSM, p_f				1			
	FP								
		FP					1		
		N	1						
		T_L	1						
		τ	1						
		p_f	1						
	RT								
		RT Defects					1		
		FSM,p_f					1		
	TC								
		FP			1				
		FP_{Miss}			1				
		LOC_{Tested}			1				
		LOC_{IMPL}			1				
		T_L			1				
		τ			1				
		FSM,p_f			1				
LOCAT									
	B/LOC								
		S_i						1	
		T_L						1	
		τ						1	
		p_f						1	
	DD								
		DD Defects					1		
		FSM, p_f					1		
	FP								
		FP					1		
		N							1
		T_L							1
		τ							1
		p_f							1
	RT								
		RT Defects						1	
		FSM,p_f						1	
	TC								
		FP			1				
		FP_{Miss}			1				
		LOC_{Tested}				1			
		LOC_{IMPL}				1			
		T_L				1			
		τ				1			

		FSM,p_f				1			
LOCAT_Frank									
	B/LOC								
		S_i			1				
		T_L			1				
		τ			1				
		p_f			1				
	DD								
		DD Defects					1		
		FSM, p_f					1		
	FP								
		FP						1	
		N						1	
		T_L						1	
		τ						1	
		p_f						1	
	RT								
		RT Defects						1	
		FSM,p_f						1	
	TC								
		FP							1
		FP_{Miss}							1
		LOC_{Tested}							1
		LOC_{IMPL}							1
		T_L							1
		τ							1
		FSM,p_f							1
LOCAT_James									
	B/LOC								
		S_i					1		
		T_L					1		
		τ					1		
		p_f					1		
	DD								
		SRS,DDR	1						
		FSM, p_f	1						
	FP								
		FP		1					
		N		1					
		T_L		1					
		τ		1					
		p_f		1					
	RT								
		RT Defects					1		
		FSM,p_f					1		

	TC								
		FP							1
		FP_{Miss}							1
		LOC_{Tested}							1
		LOC_{IMPL}							1
		T_L							1
		τ							1
		FSM,p_f							1
LOCAT_Barker									
	B/LOC								
		S_i			1				
		T_L			1				
		τ			1				
		p_f			1				
	DD								
		DD Defects		1					
		FSM, p_f							1
	FP								
		FP				1			
		N				1			
		T_L				1			
		τ				1			
		p_f				1			
	RT								
		RT Defects	1						
		FSM,p_f	1						
	TC								
		FP			1				
		FP_{Miss}			1				
		LOC_{Tested}			1				
		LOC_{IMPL}			1				
		T_L						1	
		τ						1	
		FSM,p_f				1			

Table B-1 The experiment design for the measurement phase

The "1" in a column implies that the task was performed by the corresponding subject. DD defects refer to defects found for construction of the Defect Density RePS. These defects were found through requirements, design and code inspection for each application. RT defects refer to the defects found for the construction of the Requirements Traceability RePS. These defects were found by tracing the code to requirements and finding out the

requirements that were amiss. Sub1 through Sub7 refers to the seven different subjects who participated in the Measurement Phase.

Appendix C: Further Investigation on Interaction of Defects

In this section the interaction among defects is further investigated and their effect on the failure probability is analyzed. This may provide the direction of future research on the assumption of the mutual exclusiveness of defects. It also discusses how the interaction of defects may affect the modelling of the error forms.

Case1:

Let us consider a program structure as shown in the following flow diagram

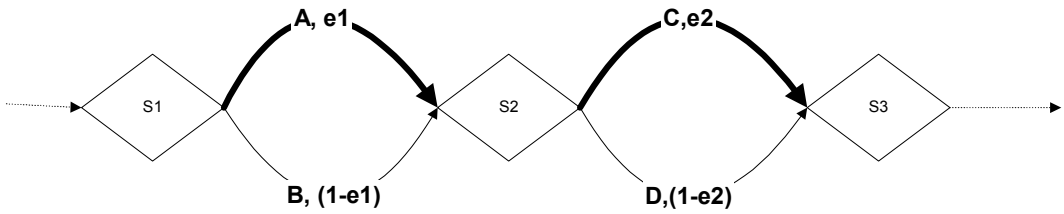


Figure C-1 Control flow graph -1

Say S1, S2, S3 are predicates and A, B, C, D, are the different paths taken. The shaded paths A and C contain a defect each, D1 and D2 respectively. Figure C-2 shows the interaction among defects.

Now, the failure probability when defects interact mutually exclusively is $k1 + k2$ where

$k1 = e1 \times I1 \times P_{D1->S2} \times e2' \times P_{S2->D2} \times P_{D2->End}$ and

$k2 = e1 \times e2 \times I2 \times P_{D2->End} + (1-e1) \times e2 \times I2 \times P_{D2->End}$. This is also the real failure probability as modeled by the simulation error form.

The failure probability with interaction of defects is $e1 \times I1 \times P_{D1->S2} \times e2' \times P_{S2->D2} \times P'_{D2->End} + e1 \times I1' \times e2 \times I2 \times P_{D2->End} + (1-e1) \times e2 \times I2 \times P_{D2->End}$. This is also the real failure probability and is equal to $k1 \times (P'_{D2->End} / P_{D2->End}) + k2 - I2 \times e1 \times e2 \times I1 \times P_{D2->End}$ which is equal to $k1 \times \alpha + k2 \times \beta$ where

$\alpha = P'_{D2->End} / P_{D2->End}$ and $\beta = 1-e1 \times I1$.

Please note that β is always less than one.

Therefore the difference between modeled failure probability and the real failure probability is $k1 + k2 - k1 \times \alpha - k2 \times \beta$. This will be a conservative estimate if α is less than one.

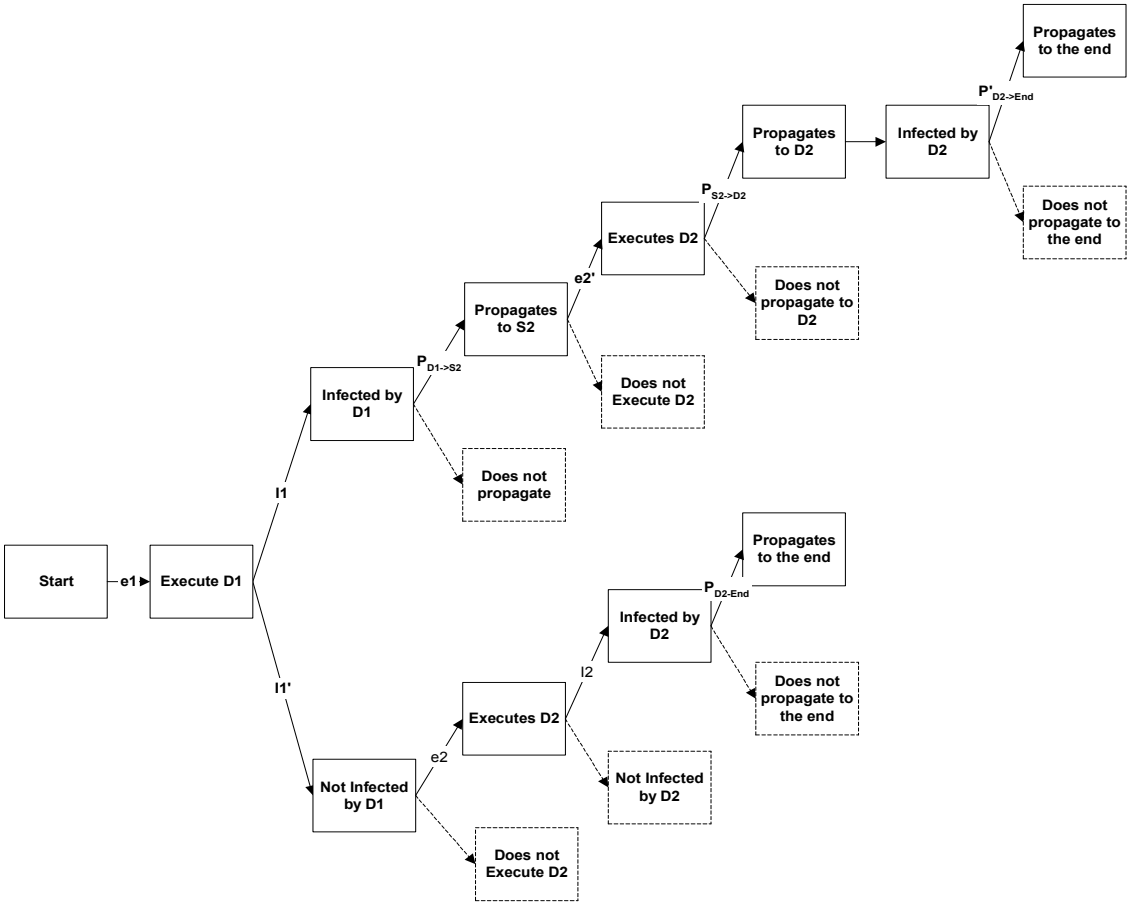


Figure C-2 Flow Graph showing interaction among defects

In this case if both the defects are detected, the modeled error is *k1 + k2 – (k1* + k2*)* and the actual error is k1× α + *k2*× β - *(*k1*× α * + *k2**× β **).* If one of the defects (D1 say) is detected the actual error in the estimate is k1× α + *k2*× β - *k1* where as the modeled error is k1 + k2 – k1*.

Case 2:

Now consider the following:

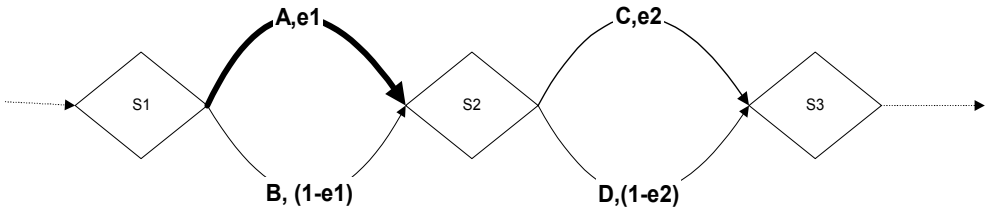


Figure C-3 Control flow graph – 2

Let us assume that there are two defects on the path A, D1 and D2. Figure C-4 shows the interaction among defects.

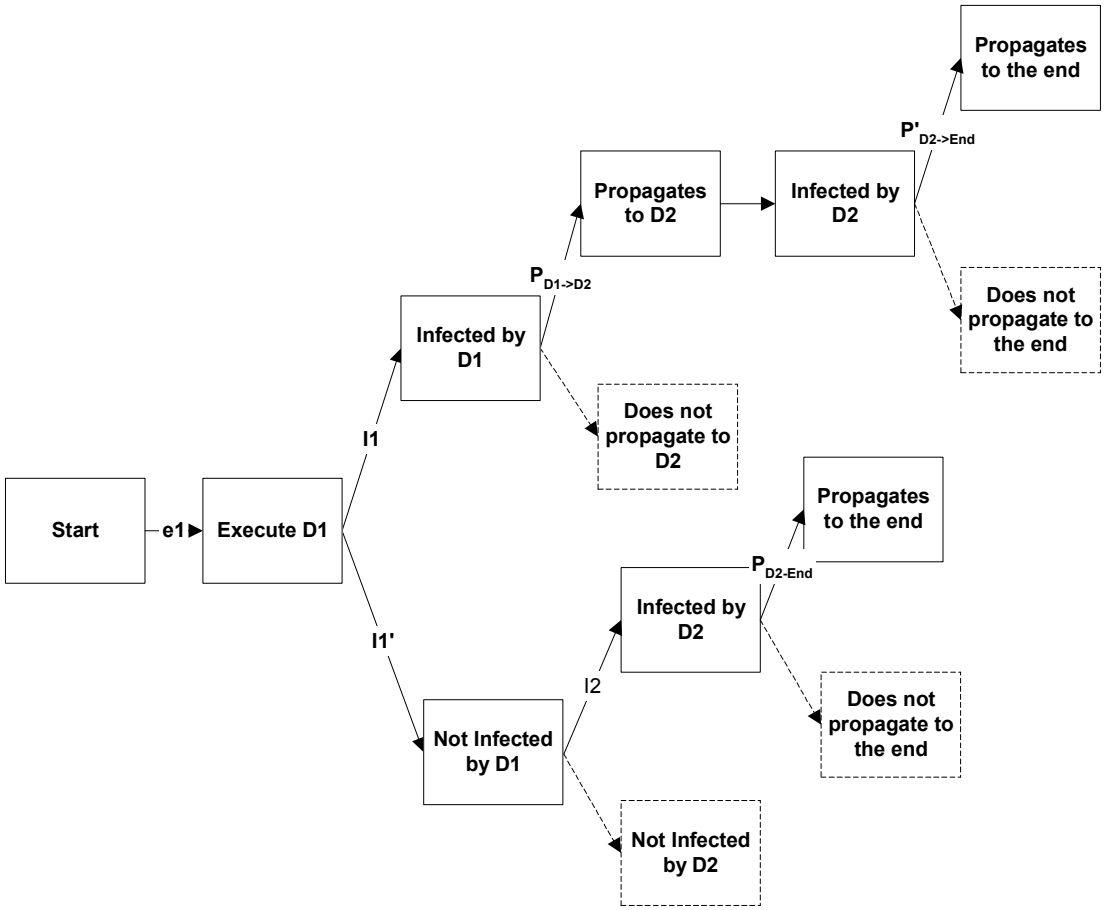


Figure C-4 Flow Graph showing interaction among defects

Now, the failure probability when defects interact independently is $k1 + k2$ where

$k1 = e1 \times I1 \times P_{D1->D2} \times P_{D2->End}$ and $k2 = e1 \times I2 \times P_{D2->End}$. This is also the real failure probability as modeled by the simulation error form.

The failure probability with interaction of defects is $e1 \times I1 \times P_{D1->D2} \times P'_{D2->End} + e1 \times I1' \times I2 \times P_{D2->End}$. This is also the real failure probability. This is equal to $k1 \times \alpha + k2 \times \beta'$ where $\beta' = 1 - I1'$. Therefore the difference between modeled failure

probability and the real failure probability is $k1 + k2 - k1 \times \alpha - k2 \times \beta'$. This will be a conservative estimate if α is less than one. β' is always less than one.

In this case if both the defects are detected, the modeled error is k1 + k2 – (k1* + k2*) and the actual error is $k1 \times \alpha + k2 \times \beta' - (k1^* \times \alpha^* + k2^* \times \beta'^*)$. If one of the defects (D1 say) is detected the actual error in the estimate is $k1 \times \alpha + k2 \times \beta' - k1$ where as the modeled error is $k1 + k2 - k1^*$.

Case3:

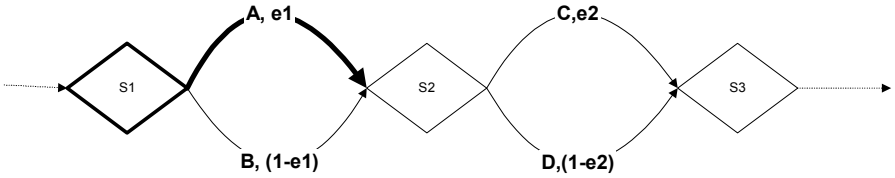


Figure C-5 Control Flow Graph – 3

Let us assume that there are two defects, on the predicate S1 and on the path A. If the defect in S1 affects the path A only, it will be equivalent to two defects on path A and the failure probability can be calculated as in Case 2. If the defect affects only path B, the case is equivalent to two defects that are mutually exclusive and the real failure probability is the sum of the failure probabilities of the defects in Path A and Path B. If the defect affects both the branches, it will be equivalent to two defects in A (D1 and D2, say) and one defect in path B (say D3). Now the real failure probability is $k1 \times \alpha + k2 \times \beta' + k3$, where, $k1$, α, $k2$, β', are the same as in case 2 and $k3$ is the fault exposure probability of defect $D3$. The modeled failure

probability is $k1+k2+k3$. Therefore the difference between modeled failure probability and the real failure probability is $k1 + k2 - k1 \times \alpha - k2 \times \beta'$ as in Case 2.

Case 4:

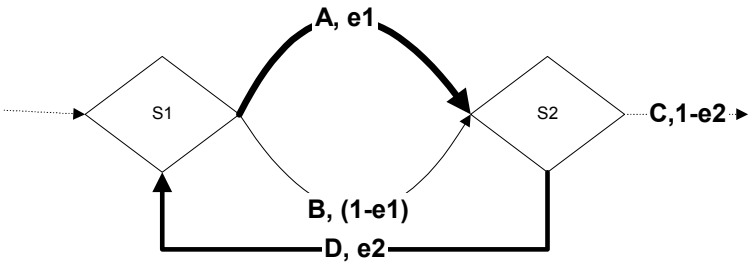


Figure C -6 Control Flow Graph – 4

The different paths of execution are S1-A-S2-End, S1-B-S2-End, S1-A-S2-D-S1-A-S2-End, S1-A-S2-D-S1-B-S2-End, S1-B-S2-D-S1-A-S2-End, and S1-B-S2-D-S1-B-S2-End. The paths of execution with interaction of defects are S1-A-S2-D-S1-A-S2-End, S1-A-S2-D-S1-B-S2-End, S1-B-S2-D-S1-A-S2-End and S1-B-S2-D-S1-B-S2-End. The assumption here is that the loop S2 to S1 is traversed at most once. Therefore the execution probability of C after the loop is traversed once, is equal to one.

Paths similar to S1-A-S2-D-S1-B-S2-End and S1-B-S2-D-S1-A-S2-End have been analyzed in Case 1. Paths similar to S1-B-S2-D-S1-B-S2-End have been analyzed in Section 4-1.

The path which remains to be studied is S1-A-S2-D-S1-A-S2-End. Figure C-7 shows the flow graph depicting the interaction of defects. The continuation of the flow graph is shown in Figure C-8.

The assumption here is that once a previous defect has propagated to a location where another defect exists, infection due to the second defect always occurs. This is a reasonable assumption because the data has already been infected and the second defect executes on the already infected data. [Malaiya92] states that infection is the probability that a change to the program causes a change in the resulting internal computational state of the program. Therefore when a defect executes on a data state that is already infected, the probability that the defect would infect the data state can reasonably be assumed to be one.

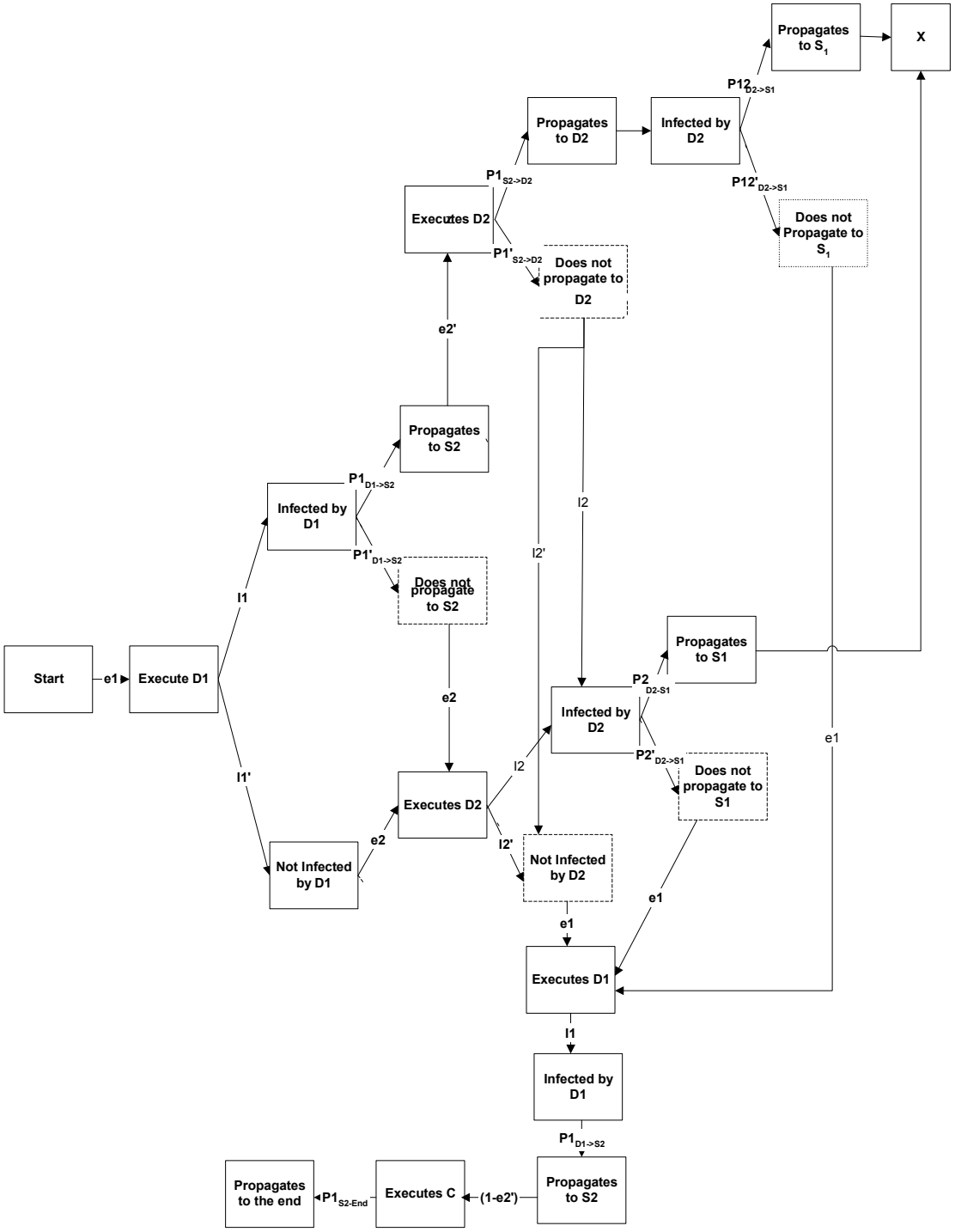


Figure C-7 Control Flow Graph showing interaction of defects

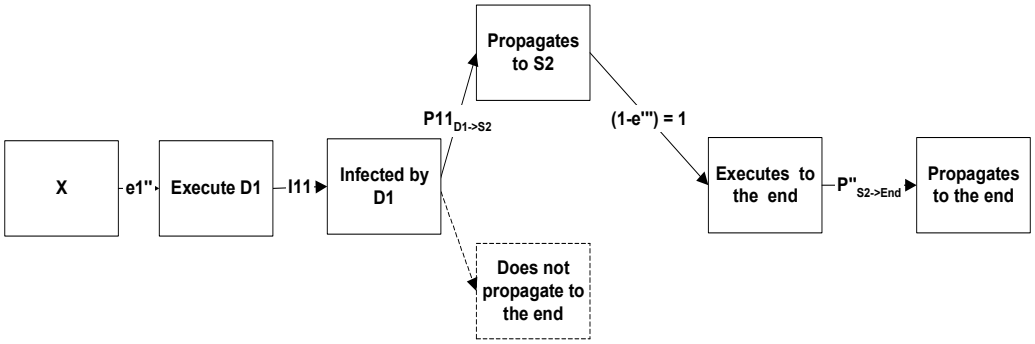


Figure C-8 Continuation of the flow graph for case 4

Now, the failure probability when defects interact independently is $k1 + k2$ where

$k1=e1\times I1\times P1_{D1->S2}\times e2'\times P1_{S2->D2}\times P1_{D2->S1}\times e1'\times P1''_{S1->D1}\times P1''_{D1->S2}\times 1\times P1_{S2->End}$ and $k2=e1\times e2\times I2\times P2_{D2->S1}\times e1'\times P2'_{S1->D1}\times P2'_{D1->S2}\times 1\times P2_{S2->End.}$ This is also the failure probability as modeled by the simulation error form.

The failure probability with interaction of defects is

$e1\times I1\times P1_{D1->S2}\times e2'\times P1_{S2->D2}\times P12_{D2->S1}\times e1''\times I11\times P11_{D1->S2}\times 1\times P''_{S2->End}+$

$e1\times I1\times P1_{D1->S2}\times e2'\times P1_{S2->D2}\times P12'_{D2->S1}\times e1\times I1\times P1_{D1->S2}\times 1\times P1_{S2->End}+$

$e1\times I1\times P1_{D1->S2}\times e2'\times P1'_{S2->D2}\times I2\times P2_{D2->S1}\times e1a''\times I1a\times P11a_{D1->S2}\times 1\times$

$Pa''_{S2->End}+$

$e1\times I1\times P1_{D1->S2}\times e2'\times P1'_{S2->D2}\times I2\times P2'_{D2->S1}\times e1b\times I1b\times P1b_{D1->S2}\times 1\times P1b_{S2->End}+$

$e1\times I1\times P1_{D1->S2}\times e2'\times P1'_{S2->D2}\times I2'\times e1c\times I1c\times P1c_{D1->S2}\times 1\times P1c_{S2->End}+$

$e1\times I1\times P1'_{D1->S2}\times e2\times I2\times P2d_{D2->S1}\times e1d''\times I1d\times P11d_{D1->S2}\times 1\times Pd''_{S2->End}+$

$e1\times I1\times P1'_{D1->S2}\times e2\times I2\times P2'_{D2->S1}\times e1f\times I1f\times P1f_{D1->S2}\times 1\times P1f_{S2->End}+$

$e1\times I1\times P1'_{D1->S2}\times e2\times I2g'\times e1g\times I1g\times P1g_{D1->S2}\times 1\times P1g_{S2->End}+$

$e1 \times I1' \times e2 \times I2h \times P2h_{D2->S1} \times e1h'' \times I11h \times P11h_{D1->S2} \times 1 \times Ph''_{S2->End} +$

$e1 \times I1' \times e2 \times I2j \times P2j'_{D2->S1} \times e1j \times I1j \times P1j_{D1->S2} \times 1 \times P1j_{S2->End} +$

$e1 \times I1' \times e2 \times I2k' \times e1k \times I1k \times P1k_{D1->S2} \times 1 \times P1k_{S2->End.}$.

As can be seen, the relationship between the above expression and the modeled failure probability (*k1 + k2)* is not directly observable. One of the avenues of future work is to investigate the interaction of defects and establish the relationship between the real and modeled failure probability. An experiment may be conducted to analyze and investigate the interaction of defects and incorporate the results into the error models.

Bibliography

[Albert01] Albert PS, McShane LM, Shih JH;.Latent *Class Modeling Approaches for Assessing Diagnostic Error without a Gold Standard: With Applications to p53 Immunohistochemical Assays in Bladder Tumors Source*: Biometrics, vol:57 iss:2 pg:610 -619, June 2001

[Anda02] Anda, B., Sjøberg, Dag I. K., *Requirements engineering: Towards an inspection technique for use case models* Proceedings of the 14th international conference on Software engineering and knowledge engineering SEKE '02 July 2002

[Apostolakis90] Apostolakis, G. (1990). "The Concept of Probability in Safety Assessments of Technological Systems." Science 250: 1359-1364.

[Apostolakis95] Apostolakis, G. (1995). "A Commentary On Model Uncertainty". In Model Uncertainty: Its Characterization and Quantification, Annapolis, Maryland, USA, October 20-22, 1993, Center for Reliability Engineering University of Maryland.

[ASFC87] *Methodology for Software Prediction*. RADC-TR-87-171. New York: Griffiss Air Force Base, 1987

[Badar05] Badar, M. Affan, Raman, S, Pulat, P. Simin *Experimental verification of manufacturing error pattern and its utilization in form tolerance sampling.* International Journal of Machine Tools & Manufacture Jan2005, Vol. 45 Issue 1, p63 , 2005

[Banker89] Banker, R. D. , and Kemerer, C. F., *Scale Economies in new software development,* IEEE Trans. Software Engineering, pp. 1199-1205, Oct. 1989.

[Bessler03] Bessler, D, Yang, J, Wongeharupan, M., *Price dynamics in the international*

wheat market: modeling with error correction and directed acyclic graphs, Journal of Regional Science, Vol. 43, pp. 1-33, 2003

[Bier95] Bier, V. M. (1995). "Some Illustrative Examples of Model Uncertainty." In Model Uncertainty: Its Characterization and Quantification, Annapolis, Maryland, USA, October 20-22, 1993, Center for Reliability Engineering University of Maryland.

[Bierman95] Bierman, G.S., *Error modeling for differential GPS*, Publisher: Charles Stark Draper Laboratory Inc., National Aeronautics and Space Administration, National Technical Information Service, 1995

[Boehm] Boehm B.W. , *Software Engineering Economics*. Prentice-Hall, New Jersey.1981.

[Brannigan93] Brannigan,V., and Meeks,C., *Computerized Fire Risk Assessment Models: A Regulatory Effectiveness Analysis,* Model Uncertainty: Its characterization and Quantification, Proceedings of the International Workshop Series on Advanced Topics in Reliability and Risk Analysis, 1993

[Brocklehurst90] Brocklehurst, S, Chan, P.Y., Littlewood, B., and Snell, J.,: Re-calibrating Software Reliability Models, IEEE Trans. Software Eng., Vol 16, No. 4, 1990, pp. 458-469

[Butler93] Butler, R. W**.** and Finelli, G. B., *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software* IEEE Transactions on Software Engineering VOL 19. NO. 1, JANUARY 1993

[Campbell63] Campbell, D.T., Stanley, J. C., *Experimental and Quasi-Experimental Designs for Research.* Chicago: Rand McNally and Company, 1963.

[Chaar93] Chaar, J.K.; Halliday, M.J.; Bhandari, I.S.; Chillarege, R. *In-process evaluation for software inspection and test,* Software Engineering, IEEE Transactions on Volume 19, Issue 11, Nov 1993 Page(s):1055 – 1070

[Chatfield96] Chatfield, C. (1996). "Model Uncertainty and Forecast Accuracy." Journal of Forecasting 15(7): 495-508.

[Chen03] Chen, J, *Using the SAS Statistical Tool from Thesis to Career: Current Experiences and Future Outlook*, Midwest SAS Users Group 14th Annual Conference, Minneapolis, MN, September 15, 2003,

[Chhiber91] Chhibber, S., G. Apostolakis, et al. (1991). "A Probabilistic Framework for the Analysis of Model Uncertainty." Process Safety and Environmental Protection 69(2): 67-75.

[Cho04] Cho, Y, Haas, Carl T. Sreenivasan, S. V. Liapi, K, *Position Error Modeling for Automated Construction Manipulators*. Journal of Construction Engineering & Management , Vol. 130 Issue 1, p50, 2004

[Cromley02] Robert G. Cromley, Richard D. Mrozinski, Jr*., Analyzing Geographic Representation Error in Capacitated Location-Allocation Modeling*

[Davis79] Davis, W. W. (1979). "Approximate Bayesian Predictive Distributions and Model Selection." J. Am. Statist. Ass. 74: 312-317.

[deFinetti72] de Finetti, B. (1972). "Probability, Induction, and Statistics." New York, Wiley.

[Douglass04] Douglass, L., Biom1, a graduate level course in Biometrics, University of Maryland, College Park, Fall 2004.

[Draper87] Draper, D., J. S. Hodges, et al. (1987). "A Research Agenda for Assessment and Propagation of Model Uncertainty", RAND.

[Draper95] Draper, D. (1995). "Assessment and Propagation of Model Uncertainty." Journal of the Royal Statistical Society, Series B 57(1): 45-97.

[Draper98] Draper, D. (1998). "On the Relationship Between Model Uncertainty and Inferential/Predictive Uncertainty."

[Droguett02] Droguett, E.L., Mosleh, A., *Methodology for the treatment of Model Uncertainty*, Center for Technology Risk Studies, University of Maryland ,April 2002

[Dunsmore01] Dunsmore, A., Roper, M., Wood, M., *Systematic object-oriented inspection — an empirical study* Proceedings of the 23rd International Conference on Software Engineering July 2001

[Ferreira 86] Ferreira, P.M. and Liu, C.R. *A contribution to the analysis and compensation of the geometric error of a machining center.* Annals of the CIRP, **35**, 259–262, 1986

[Field03] Field, A., Hole G., *How to Design and Report Experiments* .London, UK: SAGE Publications, 2003

[Fogarty96] Fogarty, M. J., Mayo, R. K. , O'Brien', L., Serchuk, F. M., Rosenberg, A. A., *Assessing uncertainty and risk in exploited marine populations, Reliability Engineering and System Safety* ,54, 183-195, 1996

[Gaffney84] Gaffney, J.E., *Estimating the Number of Faults in Code,* IEEE Transactions on Software Engineering, vol. 10, pp. 459-64, 1984

[Ghose04A] Ghose, S. *Software Requirements Specifications for LOCAT,*

[Ghose04B] Ghose,S. *Software Requirements Specifications for SSP*, University of Maryland, College Park, MD January 2004.

[Ghose04C] Ghose, S. *Software Requirements Specifications for TELLERFAST,*

[Ghose04D] Ghose, S. *Software Requirements Specification for Student Registry Query System (SRQS),* University of Maryland, College Park January 2004.

[Ghose04E] Ghose, S. , *Software Requirements Specifications for an Word Processor Unit (WPU),* University of Maryland, College Park, MD January 2004.

[Ghose04F] Ghose, S. *Software Requirements Specifications for LOCAT-I,* University of Maryland, College Park, MD January 2004.

[Ghose04G] Ghose, S. *Software Requirements Specifications for LOCAT-II,* University of Maryland, College Park, MD January 2004.

[Ghose04H] Ghose, S. *Software Requirements Specifications for LOCAT-III,* University of Maryland, College Park, MD January 2004.

[Goel78] Goel, A. L., and Okumoto, K., *An analysis of recurrent software errors in real time control systems,* Proceedings of ACM conference, pp 496-501, 1978

[Goldie03] Goldie, S. J., Kuntz, K.M., *A Potential Error in Evaluating Cancer Screening: A Comparison of 2 Approaches for Modeling Underlying Disease Progression,* Medical Decision Making 23, no. 3, 232-241, 2003

[Hatton97] Hatton, L., *Re-examining the Defect-Density versus Component Size Distribution* IEEE Software, pp. 89-97, March 1997

[Helton96A] Helton, J. C. (1996). "Probability, Conditional Probability and Complementary Cumulative Distribution Functions in Performance Assessment for

Radioactive Waste Disposal." Reliability Engineering and System Safety 54: 145-163.

[Helton96B] Helton, J. C., D. R. Anderson, et al. (1996). "Uncertainty and Sensitivity Analysis Results Obtained in the 1992 Performance Assessment for the Waste Isolation Plant." Reliability Engineering and System Safety 51: 53-100.

[Hoeting98] Hoeting, J. A., D. Madigan, et al. (1998). "Bayesian Model Averaging."Technical Report 9814, Department of Statistics, Colorado State University.

[Hughes71] Hughes, A., Grawoig, D. *Statistics: A Foundation for Analysis.* Reading, MA: Addition Wesley Publishing Company, 1971.

[IEEE84] *IEEE Guide to Software Requirements Specification,* IEEE Standard 830, 1984.

[IEEE90] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology,* IEEE Std.610.12-1990.IEEE 1990.

[IEEE98] *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software,* IEEE, New York IEEE Std 982.2, 1988.

[Jackson96] Jackson, M.R.C., Zhao,Y., Slattery, R, *Effects of modeling errors on trajectory predictions in air traffic control automation*, Guidance, Navigation and Control Conference, San Diego, CA, July 29-31, 1996

[Jelinski] Jelinski Z. and Moranda P., Software reliability research, In W. Freiberger, Ed*., Statistical Computer Performance Evaluation*, Academic, New York, 1972, pp.465-484

[Jiao04] Jiao, Y., Chen, Y., Schneider,D., and Wroblewski,J., *A simulation study of impacts of error structure on modeling stock–recruitment data using generalized linear models,* Published on the NRC Research Press Web site at http://cjfas.nrc.ca on 3 March

2004.

[Jones96] Jones, C. *Applied Software Measurement*, 2nd ed. New York: McGraw-Hill, 1996.

[Jones97] Jones, C. *Software Quality — Analysis and Guidelines for Success*. Boston, MA: International Thomson Computer Press, 1997.

[Kelly00] Kelly,D., Shepard, T., *Task-directed software inspection technique: an experiment and case study,* Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research November 2000

[Kelly03] Kelly,D., Shepard, T., *An experiment to investigate interacting versus nominal groups in software inspection.* 122-134, Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research October 2003

[Kim98] Kim, G., J., Burns, J., R., *Error reduction in distributed DSS through coordination of modeling activities: Simulation study,* Journal of Computing and Electronic Commerce, 8(3), 217-245, 1998

[Laitenberger99] Laitenberger, O., Atkinson, C., *Generalizing perspective-based inspection to handle object-oriented development artifacts*, Proceedings of the 21st international conference on Software engineering, May 1999

[Laskey95] Laskey, K. B. (1995). "Implications of Model Uncertainty for the Practice of Risk Assessment." In Model Uncertainty: Its Characterization and Quantification, Annapolis, Maryland, USA, October 20-22, 1993, Center for Reliability Engineering University of Maryland.

[Laskey96] Laskey, K. B. (1996). "Model Uncertainty: Theory and Practical

implications." IEEE Transactions on Systems, Man and Cybernetic 26(3): 340-448.

[Li93] Li, N. and Y. K. Malaiya *Enhancing accuracy of software reliability prediction*. In 4th International Symposium on Software Reliability Engineering, Denver, pp. 71—79, (1993, November).

[Li00] Li, M. and Smidts, C. Ranking Software Engineering Measures Related to Reliability Using Expert Opinion, presented at The 11th International Symposium on Software Reliability Engineering, San Jose, California, 2000.

[Li04]Li, M., Wei, Y., Desovski, D., Najad, H., Ghose, S., Cukic, B., and Smidts, C., Validation of a Methodology for Assessing Software Reliability, presented at the 15th IEEE International Symposium of Software Reliability Engineering, Saint-Malo, Bretagne, France, 2004.

[Li06] Li, M., Ghose, S., Smidts, C., Arndt, S., *Assessing Software Reliability from Software Engineering Measures,* submitted to IEEE Transactions on Software Engineering, February, 2006

[Lipow74] Lipow M., Some variations of a model for software time to failure. *TRW Systems Group*, Correspondence ML-74-2260.1.9-21, Aug.,1974.

[Littlewood73] Littlewood B. and L.Verrall J., *A Bayesian reliability growth model for computer software*, Journal of Royal Statistical Society Series C, 22(3), 332-346, (1973)

[Lockheed98A] *PACS Requirements Specification,* Lockheed Martin Corporation Inc., Gaithersburg, MD July 20 1998.

[Lockheed98B] *PACS Source Code*, Lockheed Martin Corporation Inc., Gaithersburg, MD July 28 1998.

[Lockheed98C] *PACS Test Plan,* Lockheed Martin Corporation, Gaithersburg, MD, July 28 1998

[Lyu96] Lyu, M. R., *Handbook of Software Reliability Engineering*, McGraw Hill, 1996

[Madigan94] Madigan, D. and A. E. Raftery (1994). "Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window." Journal of the American Statistical Association 89(428): 1535-1546.

[Madigan96] Madigan, D., A. E. Raftery, et al. (1996). "Bayesian Model Averaging." AAAI Workshop on Integrating Multiple Learned Models.

[Malaiya92] Malaiya, Y.K., Karunanithi, N., Verma, P., *Predictability of Software-Reliability Models,* IEEE Transactions on Reliability, **VOL.** 41, NO. **4,** DECEMBER, 1992

[Malaiya93] Malaiya, Y.K., Mayrhauser, A., Srimani, P.K.,: *An Examination of Fault Exposure Ratio.* IEEE Trans. Software Eng. 19(11): 1087-1094, 1993

[Malaiya94] Malaiya, Y. K., Li, N., Bieman, J., Karcich, R., and Skibbe, B., *The Relationship Between Test Coverage and Reliability,* Colorado State University, Colorado March 15 1994.

[Malaiya98] Malaiya,Y. K., and Denton, J., *Estimating the Number of Residual Defects*, presented at Third IEEE International High-Assurance Systems Engineering Symposium, Washington, DC, USA, 1998.

[Matsumoto88] Matsumoto, K., Inoue, K., Kikuno, T., and Torii, K., *Experimental evaluation of software reliability growth models,* in Proc. 18th Int'l. Symp. Fault-Tolerant Computing (FTCS), pp. 148–153. 1988

[Mills72] Mills H.D, On the statistical validation of computer programs, *IBM Federal Syst. Div.,* Gaithersburg MD, Rep. 72-6015,1972

[Mosleh95] Mosleh, A., N. Siu, et al. (1995). Model Uncertainty: Its Characterization and Quantification, Center for Reliability Engineering - University of Maryland.

[Mullen98A] Mullen, R.E.; *The lognormal distribution of software failure rates: application to software reliability growth modeling* The Ninth International Symposium on Software Reliability Engineering, Proceedings. Page(s):134 – 142, 4-7 Nov. 1998

[Mullen98B] Mullen,R.E.; *The lognormal distribution of software failure rates: origin and evidence* The Ninth International Symposium on Software Reliability Engineering, Proceedings. Page(s): 124-133, 4-7 Nov. 1998

[Musa75] Musa J. D., *A theory of software reliability and its application,* IEEE Trans. Software eng., SE-1, 312-327, 1975

[Musa84] Musa J.D., and Okumoto K., *A logarithmic Possion execution time model for software reliability measurement,* Proc. Int. Conf. Software Eng., Orlando, FL, 230-237 (Mar.1984).

[Musa87] Musa, J.D., Iannino, A. and Okumoto, K., *Software Reliability Measurement Prediction Application*. McGraw-Hill, 1987

[Musa98] Musa, J. D., *Software Reliability Engineering - More Reliable Software, Faster Development and Testing*. New York: McGraw-Hill, 1998.

[Nelson78] Nelson E., Estimating software reliability from test data, *Microelectron. Rel,* 17, 67-74 (1978).

[NRC96] NRC, "REGULATORY GUIDE 1.152: *Criteria for Digital Computers in*

Safety Systems of Nuclear Power Plants, U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory, Washington D.C. September 1996.

[NRC97] NRC, *Regulatory Guide 1.172, Software Requirements Specifications for Digital Computer Software Used in Safety Systems of Nuclear Power Plants,* U.S. Nuclear Regulatory Commission, Office of Nuclear Regulatory, Washington D.C. September 1997.

[Pearson72] Pearson, A. V., and Hartley, H. O., *Biometrica Tables for Statisticians, Vol 2*, Cambridge, England, Cambridge University Press, 1972

[Porter94] Porter, A. A. , Votta, L. G., *An experiment to assess different defect detection methods for software requirements inspections*, Proceedings of the 16th international conference on Software engineering, p.103-112, Sorrento, Italy, May 16-21, 1994

[Porter95A] Porter, A. A. , Votta, L. G., Basili, V.R., *Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment,* IEEE Transactions on Software Engineering, v.21 n.6, p.563-575, June 1995

[Porter95B] Porter, A., Siy, H., Toman, C. A., Votta, L. G., *An experiment to assess the cost-benefits of code inspections in large scale software development*, ACM SIGSOFT Software Engineering Notes , Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering SIGSOFT '95, Volume 20 Issue 4, October 1995

[Porter98] Porter, A., Siy, H., Mockus ,A., Votta, L. G., *Understanding the sources of variation in software inspections,* ACM Transactions on Software Engineering and Methodology (TOSEM), v.7 n.1, p.41-79, Jan. 1998

[Ramamoorthy82] Ramamoorthy C.V. and Bastani F.B., Software reliability: Status and

perspectives, *IEEE Trans. Software Eng.,* SE-8, 359-371, (Jul. 1982).

[RTCA92] RTCA, *Software Considerations in Airborne Systems and Equipment Certification,* RTCA, Inc., Washington DC December 1, 1992.

[Scach93] S. R. Schach, *Software Engineering*, 2nd ed. Homewood, IL: Aksen Associates Inc., 1993.

[Schultschik97] Schultschik, R. *The components of the volumetric accuracy*. Annals of the CIRP, **26**, 223–228, 1977

[Shapiro65] Shapiro, S. S. and Wilk, M. B. "*An analysis of variance test for normality (complete samples)*", Biometrika, 52, 3 and 4, pages 591-611, 1965

[Siu85] Siu, N. and G. Apostolakis (1985). "On the Quantification of Modeling Uncertainties." 8th Intl. Conf. On Structural Mechanics in Reactor Technology, Brussels, Belgium.

[Siu92] Siu, N. O., D. Karydas, et al. (1992). Bayesian Assessment of Modeling Uncertainties; Application to Fire Risk Assessment. Analysis and Management of Uncertainty: Theory and Applications. B. M. Ayyub, M. M. Gupta and L. N. Kanal, Elsevier Science Publishers: 351- 361.

[Smidts00] Smidts, C., Li, M. , *Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems*, University of Maryland, Washington D.C. NUREG/GR-0019, November 2000.

[Smidts02] Smidts, C., Li, B., Li, M., and Li, Z., *Software Reliability Models,* in Encyclopedia of Software Engineering, vol. 2, J. J. Marciniak, Ed., 2nd ed. New York: John Wiley & Sons Inc., 2002, pp. 1594-1610.

[Smidts04] Smidts, C., Li, M., *Validation of A Methodology for Assessing Software Quality,*" Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, Washington DC NUREG/CR-6848, 2004.

[Snedecor89] Snedecor, George W. and Cochran, William G. (1989), *Statistical Methods, Eighth Edition*, Iowa State University Press.

[Sullivan91] Sullivan, M., and Chillarege. R., *Software defects and their impact on system availability -- a study of field failures in operating systems*, In Proc. 21st International Symposium on Fault-Tolerant Computing, Montreal, Canada, 1991.

[Testmaster99] *Test Master User's Guide*, Release 1.9.5, Empirix Inc., New Hampshire, 1999.

[Thompson93] Thompson, M. C., Richardson, D. J., and Clarke, L. A., An Information Flow Model of Fault Detection, presented at International Symposium on Software Testing and Analysis, Cambridge, MA, USA, 1993.

[Thompson95] Thompson, D. (1995). "The Concise Oxford Dictionary." Ninth Edition, Clarendon Press.

[Voas92] Voas, J. M., *PIE: A Dynamic Failure-Based Technique,* IEEE Transactions on Software Engineering, vol. 18, pp. 717-27, 1992.

[Wang06] Wang, H and Huang, Q, *Error cancellation modeling and its application to machining process control,* IIE Transactions, 38, 355–364, 2006

[Wang93] Wang, C. J., and Liu, M. T., *Generating Test Cases for EFSM with Given Fault Models,* presented at 12th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '93), San Francisco, CA, 1993

[Winkler96] Winkler, R. L. (1996). "Uncertainty in Probabilistic Risk Assessment." Reliability Engineering and System Safety 54(2-3): 95-111.

[WinRunner01] *WinRunner User's Guide,* Version 7.01, Mercury Interactive Inc., Sunnyvale, CA, 2001.

[Withrow90] Withrow, C., *Error density and size in Ada software*, IEEE Software, pp. 26-30, Jan. 1990.

[Wu93] Wu, K., and Malaiya, Y.K., A Correlated Faults Model for Software Reliability, Proc. IEEE Int. Symp. on Software Reliability Engineering, Nov. 1993, pp. 80-89.

[Yamada83] Yamada, S., Ohba, M., and Osaki, S., *S-Shaped reliability growth modeling for Software Error detection,* IEEE Trans.on Reliability, vol. 32, pp. 475-478, 1983.

[Zio96] Zio, E. and G. E. Apostolakis "Two Methods for the Structured Assessment of Model Uncertainty by Experts in Performance Assessments of Radioactive Waste Repositories." Reliability Engineering and System Safety 54: 225-241, 1996