Volume 4, Issue 8, August 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Classification of Software Reliability Models

Pawan Kumar Chaurasia
Department of Information Technology,
Babasaheb Bhimrao Ambedkar University
Lucknow (U.P), India

Abstract— It is the requirement of digital computer to execute system in a reliable environment for critical, real time control application. It is used to measure, planning and controlling the usage during the development, so that quality software can be produced. Reliability analysis is executed at several stages during the process of engineering. Various analytical models are proposed from last two to three decades. In this paper, is an effort to classify the various software reliability models with attributes, present a hierarchy of models and categorize the model with various features.

Keywords - Time Between Failure Model, Fault Count Model, Fault Seeding Model, Random Time Model, Fixed Time Model, Input Domain Based Model, Non Homogeneous Poisson Process Model [NHPP].

I. INTRODUCTION

Today, every field of computer and economy is depending on the software. Requirements of computer and dependency have increased the problems and failures also. From last two-three decades, size and complexity of computer has also been changed. With these changes, computers have become an essential element of any critical system. These systems depend on the reliability, usage and operation of software. Today, most of the projects executed on highly complex hardware/software in DRDO, Defense and telecommunication industry are based on real time, artificial intelligence and critical system. Most of the software systems and packages are used, distributed and installed in identical manner, which are vulnerable, cause the failure of the software system.

One of the most important attributes of software reliability engineering is 'reliability'. It is to identify the customer satisfaction factors regarding performance, usability, efficiency, portability, functionality, capability, maintainability and documentation etc. Software reliability engineering played an important role in the quantitative measurement / assessment of software quality. Software reliability engineering is defined as the *probability of failure free operation for a specified period of time, in specified environment used by the user*. *"Reliability is the capability of the software product to maintain a specified level of performance when used under specified conditions" defined by the ISO9126*. It is not easy to estimate software reliability, because there are a number of software reliability estimation models that can be used to test and analysis of failure data during software testing.

This paper presents taxonomy of the software reliability models. This study compiles different models and enlightens the various features and dimensions of reliability models. Categorization of the model is based on different dimensions. This paper is divided into four sections: introduction, definition, description and conclusion. Section II defines software reliability definitions and characteristics of software failure. Section III, categorize the different software reliability models and tabulated with features. Finally, paper is concluded in Section IV.

II. CHARACTERISTICS OF SOFTWARE RELIABILITY

Software reliability is a key factor for software quality. Software and hardware reliability both have different mechanism and parameters to identify the cause of failure. Hardware faults are physical faults while in software, it comes from design. Hardware faults are easily identified and correct, while software faults are hidden, which are harder to identify, correct, detect and classify. There are different characteristics of software failure [1][2] which are described here.

(a) When the software is implemented, software faults are occurring. These faults may remain hidden during the entire testing. Faults are identified during the correction and removes to improve the reliability of the software.
(b) Software faults are static; they existed at the time of the design of the software and remove at the time of testing.
(c) Physical and social environment factors that affect the software reliability of hardware and software like clock speed, memory utilize and memory leakage etc.
(d) Software reliability cannot be predicted from the successful execution of the software and hardware. It depends on input or from the human error and behavior.
(e) Software failure depends on the correction and identification of faults and cause of failure in the software.
(f) If the hidden bugs are not identified and corrected in the software; can cause faults or failure of the system.
(g) Reliability cannot be increased by redundancy but improved by diversity.

Figure 1 shows the various attributes of software reliability model which cover the software reliability estimation of the software. There are various attributes of software reliability estimation. In this paper, six dimensions are considered for software reliability is; Time Between Failure, Fault Count, Input Domain, Fault Seeding, Finite and Infinite Failure and Error Behavior. These attributes are considered to implement various software reliability models.

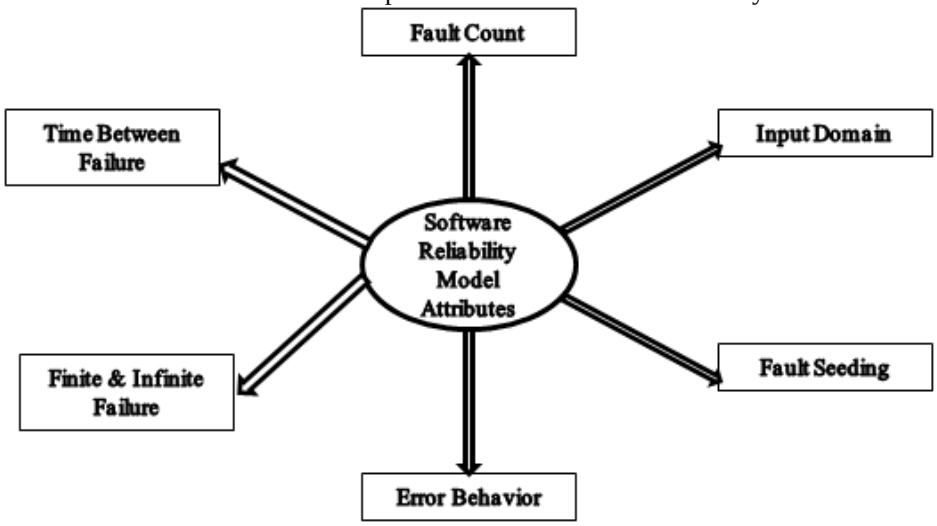

Figure 1: Attributes of Software Reliability Model

III. SOFTWARE RELIABILITY MODELS

Software reliability is one of the important attributes which improves software quality. Most of the researcher used binary (0 and 1) or yes/no to quantifies the quality of software. When software is shipped, it is found to test 100% execute and found zero error. Then the system is reliable, and ready to shift. Reliability cannot be specified on the number of errors or zero bugs/faults. To measure the reliability, test cases and methodology are proposed / implemented for various software reliability models. From last three-four decades, diverse reliability models are proposed on different parameters, techniques and methodology. It is difficult to categorize all the models on the base of its features. In this paper, author tried to accumulate models and categorize on the basis of its features and classify on different levels [3].

In figure 2 hierarchy of software reliability models are presented at initial stage, which is based on failure history and data requirements. Failure history is divided into four parts on its failure data: TBF (Time Between Failure Models), FC (Fault Count Models), Fault Seeding Models and Input Domain Models. When data are analyzed, it is divided into two parts including static and dynamic models. Further static model is divided into two parts on the base of error and input data are known as error domain models and data domain models. Data domain is divided into two parts namely RTM (Random Time Model) and FTM (Fixed Time Model). This classification is represented through hierarchical diagram [4].

Some of the important attributes are discussed in the following section.

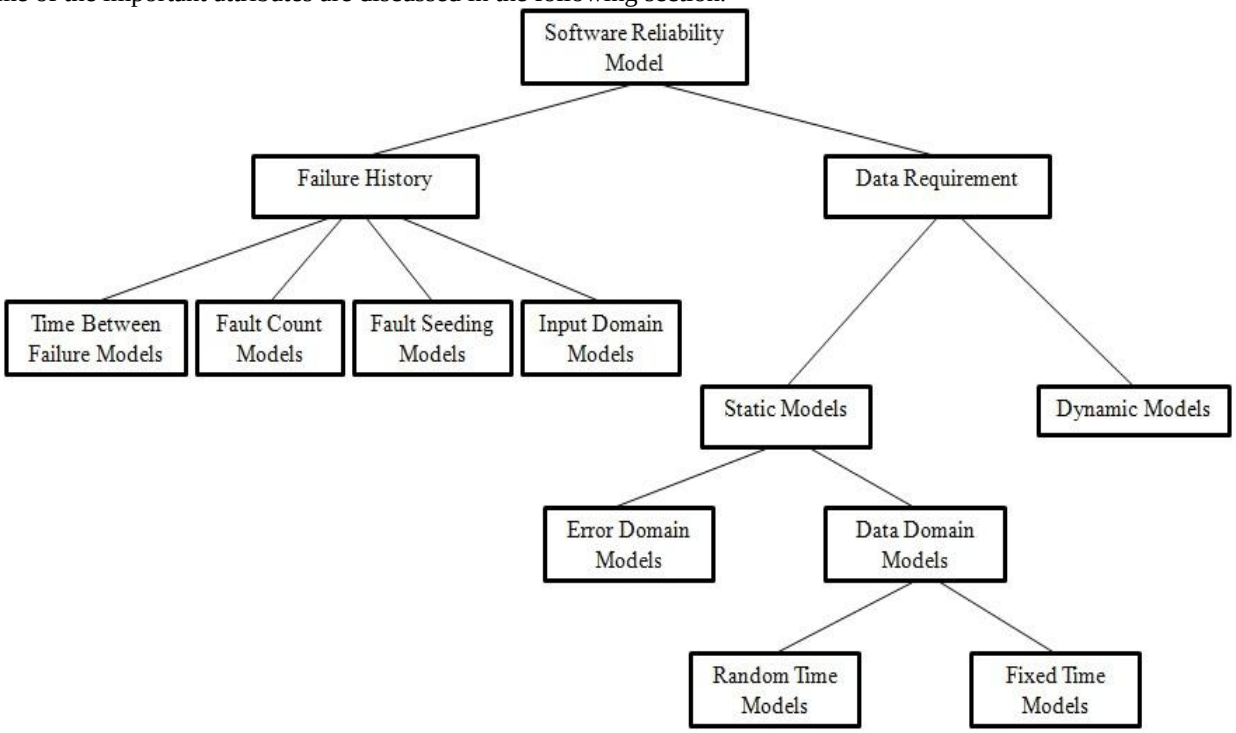

Figure 2: Hierarchy of Software Reliability Models

1. TIME BETWEEN FAILURE MODELS:

It is one of the earliest classes of proposed model for software reliability estimation. Successive failure in the system for longer period causes failure of the system. In this model time between failures is assumed for the process. The data which are considered for analysis is the time between i^{th} and $(i-1)^{th}$ failures. Fault data parameters are estimated from observed values of the times between failures and estimates of software reliability, mean time to the next failure. Some of the pertinent models under this category are briefly described in the following section.

A. Jelinski-Moranda Model:

In 1972, the most common approach used model for estimating the software reliability [5]. If there are N number of faults at the start of testing, each is independent from others and are equally cause of failure during testing. When the faults are observed, it is removed with certainty at a negligible time. New faults are not introduced during the debugging process. The current fault content of the program is calculated by using the software failure rate or the hazard functions. In other words, the hazard function is calculated during t_i time between the $(i-1)^{st}$ and i^{th} failure is given by:

$$Z(t_i) = ø\,[N-(i-1)]$$

In a function ø is used as proportionality constant. This hazard function is constant between failures, but decreases in size k shows the removal of each fault. A variation of the above model was proposed by Moranda [6]. These faults are not removed until the fatal one at which the accumulated group of faults is removed.

B. Schick and Wolverton (SW) Model:

It is based on the assumptions of the JM model except that the hazard function is assumed to be proportional to the current fault content of the program as well as to the time elapsed since the last failure[7] is given:

$$Z(t_i) = ø\,[N-(i-1)]\,t_i$$

Goel [8] define the cumulative time between the beginning of the testing t_i. Alteration of the above model was proposed in [9] whereby the hazard function is assumed to be parabolic in test time and is given by the function:

$$Z(t_i) = ø\,[N-(i-1)]\,(-at_i^2+bt_i+c)$$

In function a, b and c are defined as a constant. This function consists of two components, hazard function of the JM model and the superimposition of the second term . It indicates that the failure occurring increases rapidly as the test time accumulates within a testing interval. At failures time (t_i=0), the hazard function is proportional to that of the JM model.

C. Goel and Okumoto Imperfect Debugging Model:

When faults are detected, it is removed with certainty from the software. But it is not easy to detect and remove the faults[10]. To overcome this problem, Goel and Okumoto[11][12] proposed an imperfect debugging model, which is the extension of the JM model. In this model, the number of faults in the system at time t, X(t), is treated as a Markov process whose transaction probability are governed by the probability of imperfect debugging. Time between the transaction of X(t) are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the $(i-1)^{st}$ and the i^{th} failure is given by the function:

$$Z(t_i) = ø\,[N-p(i-1)]\,\lambda$$

Where N is the initial fault content of the system, p is the probability of imperfect debugging and λ is the failure rate per fault.

D. Littlewood-Verrall Bayseian Model:

It is not mandatory that software reliability be specified in terms of number of errors in the program. In [13][14], Littlewood and Verall discussed that the times between failures are assumed to follow an exponential distribution but the parameters of this distribution are treated as a random variable with a gamma distribution like:

$$F(t_i|\,\lambda_i) = \lambda_i\ e^{-\lambda iti}$$

From the above model it is found that the failure data in different environments cannot be defined in the LV Bayesian model.

2. FAULT COUNT MODELS:

In this model, the number of faults is counted in specified time intervals rather than time between failure. When faults are removed from the system, the number of observed failures will decrease. In this test, the time interval is fixed in advance and the number of failure values is treated as a random variable. Parameters of the failure rate can be estimated from the observed value of failure count or from failure times. Software reliability estimate means are calculated from the next failure.

The earliest model was proposed by the Shooman [15][16] for the fault count model. The same approach is used by the Musa [17], later made some amendment and proposed another fault count model, based on execution. Goel-Okumoto introduced [10] about time dependent failure rate under NHPP and the generalized of the model was proposed by Goel [18]. Some other models which are described here.

A. Musa Execution Time Model:

Musa [19] assumes that, it is similar to JM model except that the number of failures in specified execution time interval. The hazard function for this model is provided as:

$$Z(\tau) = \phi\, f(N-n_c)$$

where τ is the execution time, utilized in executing in the program, f is the linear execution frequency, ϕ is a proportionally constant fault exposure ratio that relates the fault exposure frequency to the linear execution frequency and n_c is the number of faults corrected during 0 to τ. Musa also provides a systematic approach for converting the model into calendar time.

B. Goel-Okumoto Non-Homogeneous Poisson Process (NHPP) Model:

It is assumed that a software system is failure at random time interval caused by a fault in the system [20]. Suppose N(t) is the cumulative number of failures identified in time t. Author proposed that N(t) can be modeled as a non-homogeneous Poisson process model for the following.

$$P\{N(t)=y\}=\frac{(m(t))^y}{y!} e^{-m(t)},\ y=0,1,2..$$

$$m(t) = a(1- e^{-bt})$$

$$\lambda(t) \equiv m'(t) = abe^{-bt}$$

Where m(t) is the expected number of failures observed by time t, $\lambda(t)$ is the failure rate, a is the expected number of failures to be observed and b is the fault detection rate per fault. The number of faults to be detected is treated as the random variable. The number of faults is to be fixed in the equation.

C. Goel Generalized Nonhomogeneous Poisson Process (NHPP) Model:

Goel supposed that when testing is processed, software quality automatically improves. But in practice, he observed that in many testing situations, failure rate first increases and then decreases. Goel proposed [18][21] the following generalized NHPP model.

$$N(t)=y\}=\frac{(m(t))^y}{y!} e^{-m(t)},\ y=0,1,2..$$

$$m(t) = a(1- e^{-bt})$$

Where a is expected number of faults, b and c are constants that reflect the quality of testing. The failure rate of the models is provided by the equation:

$$\lambda(t) \equiv m'(t) = abe^{-btc}\ t^{c-1}$$

D. IBM Binomial and Poisson Model:

In 1980, Brooks and Motley[22] supposed the fault detecting process during testing in a discrete process interval of time for a Poisson and Binomial Distribution. The software system is developed and tested incrementally with both the models. They arrogate that both models can be applied on system level or module level.

E. Shooman Exponential Model:

It is similar to the JM model. The hazard function for this model [15][16] perform in a following manner:

$$Z(t) = k[(N/I) - n_c(\tau)]$$

Where t is the operating time of the system, I is the total number of the instruction in the program, debugging time is τ, the total number of the faults corrected during debugging is $n_c(\tau)$ and k is the proportionally constant.

F. Generalized Poisson model:

In this model, the variation between NHPP model and Goel-Okumoto which assumes as a mean value function [23] in a following manner.

$$m(t_i) = ø\ [N - M_{i-1})]\ t_i^{\ \alpha}$$

where the total number of faults is M_{i-1} at the debugging time interval, ø and α are constant respectively for proportionality and rescaling time.

G. Musa-Okumoto Logarithmic Poisson Execution Time Model:

It is similar to the Goel-Okumoto model with a mean value function of α. In this model [24], the observed number of failures is assumed to be NHPP model which is defined in the function as:

$$\mu(\tau) = 1/\theta.\ Ln\ (\lambda_0\ \theta_\tau + 1)$$

where initial failure intensity and reduction in the normalized failure intensity defined by λ_0 and θ respectively. This model is also closely related to Moranda's geometric de-eutrophication model [25].

3. FAULT SEEDING MODELS:

In this model, author 'seeded' a known number of faults in a program. It is assumed to have an unknown number of indigenous faults in the program. Indigenous and seeded faults are counted and identified during the testing. Using different methodology and models, the number of indigenous faults in the program and the reliability of the software are estimated.

In this category, Mills Seeding Model are implemented at the initial stage for seeding the faults, later Basin and Lipow [28] also introduced related to some models on the base of seeding methodology.

A. Mills Seeding Model:

It is one of the famous fault seeding model [26]. It required a random number of faults are seeded in a program to be tested. The program is then tested for some interval. The number of original indigenous faults can be estimated from the number of indigenous faults and seeded faults which are not covered at the time of testing. These models are also known as a tagging model since a given fault is tagged as seeded or indigenous.

B. Basin Model:

Basin [27], proposed a two stage procedure with the use of two programmers which can be used to estimate the number of indigenous faults in the program.

C. Lipow Model:

Lipow [28], consider the problem for identify a fault of any kind in any test of the software. Then identified the probability of finding given number of indigenous and seeded faults are calculated for independent tests.

4. INPUT DOMAIN BASED MODELS:

The fundamental approach in the input domain is to generate the test cases from an input domain. It is difficult to partition the input domain into equivalence classes. The reliability measure is calculated from the number of failures or execution of the test cases. For input domain, Nelson[29] and Ramamurthy and Bastin Model[30] are introduced.

A. Nelson Model:

Nelson proposed a model [29], to measure the reliability of software for the sample of n inputs. Inputs are randomly chosen from the input domain. Random sampling of n inputs is done according to a probability distribution and the set is the operational profile or simply the user input distribution.

The difference between Nelson's R1, Brown and Lipow's R2 is to incorporate the usage distribution or test case distribution, while the later implicitly assumes that the accomplished testing is represented of the expected usage distribution. In both models have prior knowledge of the operational usage distribution.

B. Ramamurthy and Bastani Model:

In this input domain based model, author are considered with the reliability about the real time, critical process and control programs. In this system, no failure should be detected during the reliability estimation phase, so that the reliability is one. This model provides an estimate of the conditional probability that the program is correct for all possible inputs and it is correct for a specified set of inputs.

5. DATA DOMAIN MODELS:

These are the static models, developed for different sets of inputs for observed software failures. It is subdivided into two parts: random time model and fixed time model. It is described by Nelson [29] in 1973, for software reliability models.

A. RANDOM TIME MODELS:

In this model, the number of errors is detected during random time interval. The number of failures is calculated in the random time interval. Number of data are collected is called the length of the each interval. Shooman [15][16] and Lapadulla [31] are famous model in this category.

B. FIXED TIME MODELS:

These are the static models where failure data are fixed time interval of equal length. These models assume for different type intervals. Each error is detected independently for different type of intervals. Schneidwind NHPP and Moranda Geometric Poisson models [32][25] fall in this category.

Table 1: Software Reliability Models

Models Category	Proposed Models	Features	Year	Reference
TBF (Time Between Failure Models)	JM De- Eutrophication Model	• Each fault is independent of others. • Detected fault is removed with omit time and no new faults are attached during the process.	1972	[5]
	Schick and Wolverton(SW) Model	• Hazard function is linear with time within each failure rate interval. • It is to be assumed as a parabolic in test time.	1978	[7][8][9]
	Goel & Okumoto Imperfect Debugging Model	• It is the extension of JM model. • The number of faults in the system is treated as a markov process.	1978	[10][11][12]
	Littlewood-Verrall Bayesian Model	• Reliability should not be specified in terms of the numbers of errors in the program. • The failure parameter cannot be defined by this model.	1973	[13][14]
FC (Fault Count Models)	Shooman Model	• Faults are corrected during execution.	1972	[15][16]
	Goel-Okumoto NHPP Model	• Failure at random-time caused by faults.	1979	[10]
	Goel Generalized NHPP Model	• To observe the Increasing / decreasing failure rate process.	1982 1983	[20]
	IBM Binomial and Poisson Models	• Fault detection process during software testing to be a discrete process.	1980	[22]
	Musa-Okumoto Logarthimic Poisson Execution Time Model	• Observed no of failures after time t is assumed to be a NHPP, similar to the GO model.	1984	[24]
	Musa Execution Time Model	• No of failures is specified in execution time intervals. • It convert the model, so that it is applicable for calendar time.	1971	[18][19]
FS (Fault Seeding Models)	Lipow Model	• Finding a fault of any kind in any test of the software. • Probability of indigenous faults in the program	2000	[28]
	Basin Model	• Estimate indigenous faults in the program.	1974	[27]
	Mills Seeding Model	• No of known faults be randomly seeded in the program to be tested.	1972	[26]
IDB (Input Domain Based Models)	Nelson Model	• Inputs are randomly chosen from the input domain. • Prior knowledge of the operational usage distribution.	1978	[29]
	Ramamoorthy and Bastani Model	• Reliability estimate for real time and critical time process control programs. • Estimate of the conditional probability for all possible inputs.	1982	[30]
DDM (Data Domain	Nelson Model	• Measure reliability by running the software for a sample of n random inputs.	1973	[29]

Models)				
RT (Random Time Models)	Shooman Exponential Model	• Hazard function, operating time of the system measured from its initial activation.	1972	[15][16]
	Lapadulla Model	• Data are used with discrete time, random time interval.	1976	[31]
FT (Fixed Time Models)	Schneidewing Model	• Number of failures as per intervals, with all time is of equal length.	1975	[32]
	Moranda Geometric Poisson Model	• The number of failures during specified time intervals as being independent Poisson Random variables.	1975	[25]

IV. CONCLUSION

Today people depend on computer, which increases the reliability problem. From last two three decades several incidents have been occurred. These problems take place, when faults are encountered during the execution of the program. Software failure occurs, if the behavior of the software differs from actual performance of the software. This study establishes the fact about the traditional software reliability models. To estimate reliability, count non-error models which are helpful to identify new errors. It is also helpful to identify the remaining faults by error seeding technique models.

In this paper, author compiled and reviewed various software reliability models. In the above discussion, few assumptions are evaluated, time between failures are independent of each other, detected fault is immediately removed, before proceeding to the nest phase of the system, no new faults are introduced during the fault removal process, failure rate is decreases with test time and is proportional to the number of remaining faults. All these models are classified into different classes. Author attempted to point-out the features of different models in a tabular form which is helpful for researcher in analysis. Above analytical models are primarily useful in estimating and monitoring viewed as a measure of software reliability estimation and improve quality.

REFERENCES

[1] S.J. Keene, "Comparing Hardware and Software Reliability, Reliability Review", Vol. 14. No 4, December 1994, page 5-21.

[2] D.S. Herrmann, "Software Safety and Reliability", IEEE Computer Society Press, Los Alamitos, 1999.

[3] A. Yadav & R.A. Khan, "Critical Review on Software Reliability Models", IJRTE, Vol 2, NO 3, November 2009.

[4] L. Shanmugam & L. Florence, "An Overview of Software reliability models", IJARCSSE, Vol. 2, Issue 10, October 2012, pp 36-42.

[5] Z. Jelinski & P. Moranda, "Software Reliability Research", In Statistical Computer Performance Evaluation, W. Freiberger, Ed. New York: Academic, 1972, pp 465-484.

[6] P.B Moranda, "Predicting of software reliability during debugging", in Proceeding Annual Reliability and Maintainability, Symposium, Washington DC, Jan 1975, pp 327-332.

[7] G.J Schick & R.W. Wolverton, "An analysis of component software reliability models", IEEE transaction Software Engineering, Vol. SE-R, July 1982, pp 359-371.

[8] A. L. Goel, "A summary of the discussion on an analysis of competing software reliability models", IEEE Transaction Soft. Eng. Vol. SE-6, pp. 501-502, 1980.

[9] G.J. Schick & R.W. Wolverton, "Analysis of competing software reliability models" IEEE Transaction on Software Engineering, Vol. SE-4, No 2, March 1978, pg 104-120.

[10] T.A. Thayer, M. Lipow and E.C. Nelson, "Software reliability study", Rep. RADC-TR-76, 238 August 1976.

[11] A.L Goel and K. Okumoto, "An analysis of recurrent software failure in a real-time control system", in Proc. ACM Annual Tech. Conf. ACM Washington, DC 1978, pp 496-500.

[12] A.L.Goel, "A Markovian model for reliability and other performance measures of software systems", in proc. National computing conference, New York, Vol. 48, 1979, pp. 769-774.

[13] B.Littlewood & J. L. Verrall, "A Bayesian reliability growth model for computer software", Applied statistics Vol. 22, 1973, pp 332-346.

[14] B. Littlewood, "Theories of software reliability, How good are they and how can they be improved?" , IEEE transaction. Software engineering Vol.SE-6, 1980, pp 489-500.

[15] M.L. Shooman, "Probabilistic models for software reliability prediction", in Statistical Computer Performance Evaluation, W. Freiberger, Ed. New York:Academic, 1972, pp 485-502.

[16] M.L Shooman, "Software reliability measurement and models" in proc. Annual Reliability and Maintainability Symp., Washington, DC, Jan 1975, pp 485-591.

[17] J.D. Musa, "A theory of software reliability and its application", IEEE Transaction, software engineering, Vol. SE-1, 1971, pp 312-327.

[18] A.L. Goel, "A guidebook for software reliability assessment" Rep. RADC, TR-83, 176, Aug, 1983.

[19] J. D. Musa, "A theory of software reliability and its application" IEEE Trans. Software Reliability. Vol. SE-1, 1971, pp 323-327.

[20] A. L. Goel, "A time-dependent error-detection rate model for software and other performance measures" IEEE Transactions on Reliability, Vol. R-28, No 5, Aug-1979, pp. 206-211.

[21] A. L. Goel, "Software reliability modeling and estimation techniques" Rep. Aug-1982, RADC-TR 82-263.

[22] W. D. Brooks and R.W. Motley, "Analysis of discrete software reliability models", Rep. RADC-TR 80-84, April 1980.

[23] J. E. Angus, R.E. Schafer & A. Sukert, "Software reliability model validation" in Proc. Annu. Reliability and Maintainability Symp. San Francisco CA, Jan 1980, pp 191-193.

[24] J. D. Musa & K. Okumoto "A logarithmic Poisson execution time model for software reliability measurement." Proceedings of the International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, California 1984,pp. 230-238.

[25] P.B Moranda, "Software Reliability Prediction", Proceeding of the 6th Triennian World Congress of the International Federation of Automatic Control, 1975 pp 342-347.

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

[27] S.L Basin, "Estimation of software error rate via capture-recapture sampling", Science Application, Inc, Palo Alto, CA, 1974.

[28] M. Lipow, "Estimation of software package residual errors", TRW Redondo Beach, CA, Software Series, 1972, Rep.-SS 72-09.

[29] E. Nelson, "Estimating software reliability from test data" Micro electron, 1978, Rel. Vol. 17, pp 67-74.

[30] C. V. Ramamurthy & F. B. Bastani, "Software reliability: Satatus and perspective", IEEE Transaction. Soft. Engg. July-1982, Vol. SE-8, pp 359-371.

[31] J. McLean, " A comment on the 'basic security theorem' of Bell and LaPadula", Information Processing Letters, Vol. 20, No 2, Feb 1985, pp 67-70. [doij>10.1016/0020-0190(85)90065-1]

[32] N.F Schneidwind, "Analysis of error process in computer software", in Proc. International Conference,, Reliable Software, Los Angeles, CA, April 1975, pp 337-346.

