Important Milestones in Software Reliability Modeling*

Swapna S. Gokhale, Peter N. Marinos and Kishor S. Trivedi
Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708-0291
{ssg, pnm, kst}@ee.duke.edu
Phone: 919 - 660 - 5269
FAX: 919 - 660 - 5293

Abstract

A number of software reliability models have been proposed for assessing the reliability of a software system. In this paper, we discuss the time-domain and data-domain approaches to software reliability modeling, and classify the previously reported models into these two classes based on their underlying assumptions. The data-domain models are further classified into fault-seeding and input domain models, while the time-domain models are further classified into homogeneous Markov, non-homogeneous Markov and semi-Markov models. We present some representative models belonging to each of the classes, and then discuss the relative merits and limitations of the time and data-domain approaches.

1 Introduction and Background

The production of modern computer software is one of the most complex and unpredictable activities in industry. Software is an integral part of critical applications such as commercial avionics, banking, nuclear power generation and medical instrumentation and thus establishing the quality of software systems has become a major challenge in all software production environments. Production uncertainties also often delay the delivery of major products resulting in enormous unplanned costs. All these factors converge to a fundamental point, namely, the need for realistic modeling and accurate quantification of the software production process.

Numerous analytical models for predicting reliability and fault content in a software system are available. These models attempt to capture and quantify the inherent uncertainties in a software production process. The accuracy of these models, however, varies significantly as clearly pointed out by Keiller, Littlewood et al.[1, 14] and there is no single known model that performs well in all contexts. In this paper, we present an overview of key software reliability

*This work was supported in part by the US AIR FORCE Rome Laboratory as a core project in the Center for Advanced Computing and Communication and by a contract from the Charles Stark Draper Laboratory.

modeling approaches and discuss briefly their underlying assumptions, merits and limitations.

2 What is Software Reliability ?

A number of conflicting views exist as to what software reliability is and how it should be quantified. One of the approaches parallels that of program proving whereby the program is either correct or incorrect. Software reliability in this case is binary in nature; an imperfect program has zero reliability and a perfect program has a reliability value of one. Another approach is based on program testing where a percentage of successful tests is used as the measure of program quality. Software reliability in this case is defined as the relative frequency of the times that the program performs as intended. This leads us to define software reliability as the probability of fault-free operation, provided by the software product under consideration, over a specified period of time in a specified operational environment. A common method used for predicting software reliability is by means of an analytical model. The parameters of the model are estimated from the available software failure data, and the model is then used to compute relevant measures such as software reliability, availability and release times.

Prevalent approaches to software reliability modeling are based on schemes previously utilized in hardware reliability assessment, with suitable modifications to account for the fundamental differences between hardware and software systems. The source of failures in software is due primarily to design faults, while the source of failures of hardware is predominantly physical deterioration, or the "wearing out" process. A software design defect once properly fixed, is fixed in general for all times. Software failures occur mainly when the program under consideration is exposed to an operational environment for which it was not designed or tested for. The design flaws in a program, and hence the inability of the program to function properly in all environments, gives rise to the idea of "reliable software". Software reliability is thus a measure of our confidence in the ability of a program to provide acceptable performance within a given operational environment.

In the case of well-designed hardware systems, the probability of failure due to physical deterioration is generally much greater than the probability of failure due to design flaws. Therefore, the concept of "design reliability" is not very critical, and hence not considered when dealing with the overall reliability of the hardware systems.

Generally, the reliability of hardware systems without repair degrades with time while the reliability of software systems improves with time as software design flaws are detected and corrected.

3 Classification of Software Reliability models

The popular software reliability models may be classified as data-domain and time-domain models.

- Data-domain models: They are based on the philosophy that if the set of all input combinations to a computer program are identified, then an estimate of its reliability can be obtained by exercising all the input combinations and observing the outcomes. In practice, it is not feasible to identify the set of all input combinations, and this approach is reduced to a method of selecting sample data sets representative of the expected operational usage for the purpose of estimating the residual number of faults in the software product under consideration. The data-domain models can be further classified as:

  1. Fault-seeding models: The software product, which has an unknown number of indigenous faults, in this case is "seeded" with a known number of faults, and subjected to rigorous testing. An estimate of the actual number of indigenous faults is then obtained by determining the ratio of discovered seeded faults and discovered actual faults. Mills' Hypergeometric model[17] falls in this category.

  2. Input-domain models: In case of input-domain based models, the reliability of the software is measured by exercising the software with a set of randomly chosen inputs. The ratio of the number of inputs that resulted in successful execution to the total number of inputs gives an estimate of the reliability of the software product. The model proposed by Nelson[27] belongs to this category.

- Time-domain models: They model the underlying failure process of the software under consideration, and use the observed failure history as a guideline, in order to estimate the residual number of faults in the software, and the test time required to detect them. The time-domain models can be further classified as:

  1. Homogeneous Markov Models: The models belonging to this category assume that the initial number of faults in a software product under consideration is unknown but fixed. The number of faults in the system, at any time, form the state space of a homogeneous Markov chain. The failure intensity of the software, or the transition rates of the Markov chain depend upon the number of residual faults in the software. The popular models in this class are: Jelinski-Moranda[12], Goel-Okumoto Imperfect Debugging Model[9] etc.

2. Non-Homogeneous Markov Models: These models assume the number of faults present in a software product to be a random variable most often assumed to display the behavior of a Non-Homogeneous Poisson Process (NHPP). The popular NHPP models include GO model[9], Delayed S-shaped[22], etc.

3. Semi-Markov models: This class of models assumes that the initial number of faults in the software product is unknown but fixed, and that the failure intensity of the software, or the transition rate from a given state, depends not only on the number of residual faults in the software, also on the time elapsed in that state. Schick-Wolverton[24] etc. belongs to this class.

4. Other Models: Some of these models include: Littlewood-Verall Bayesian model[1], Keiller-Littleweood model[1].

The classification of the software reliability models is shown graphically in Figure 1. The state-space view of time-domain models that we have adopted has the advantage of being easily extendible to include imperfect detection/repair and finite repair times[11].

4 Time-Domain Models

In this section, we briefly discuss the various classes of models that fall into the category of "time-domain models" with representative examples.

4.1 Homogeneous Markov Models

The models belonging to this class assume that the initial number of faults residing in the software is fixed but unknown. The number of faults remaining in the software product forms the state space of a homogeneous Markov chain. The software system is said to be in state i, if i is the number of faults remaining in the software, and the transition rate to state $(i-1)$, depends on the current state i. Thus, the sojourn time in state i, denoted by T_i, is exponentially distributed. Some of the popular models in this class are described below:

Jelinski-Moranda De-Eutrophication Model[12]

This model is credited with being the first model for assessing software reliability. It assumes that the software under consideration has N faults at the beginning of testing. Each of these faults is independent of the others and all faults are equally likely to cause a failure during testing. The repair process is assumed to be instantaneous and perfect, i.e., the time taken to remove a fault is negligible, the fault is removed with certainty and no new faults are introduced

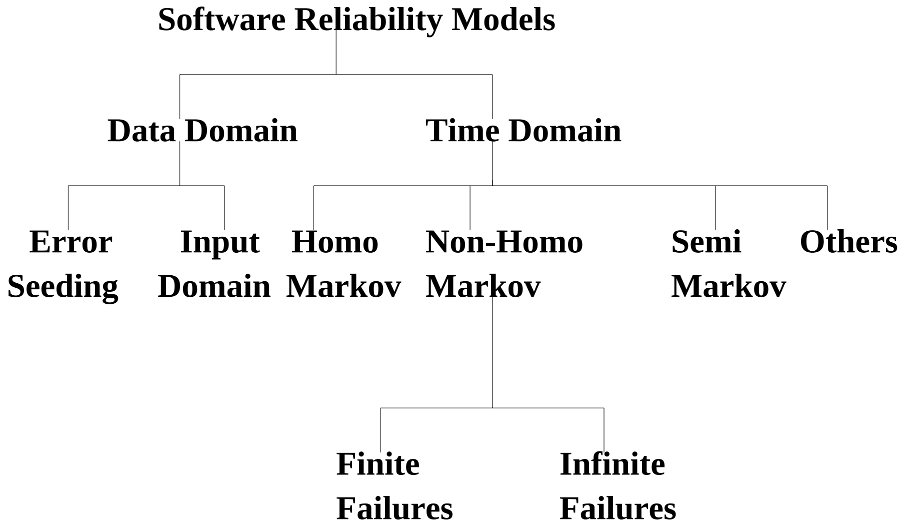

Figure 1 : Classification of Software Reliability Models

Figure 1: Classification of Software Reliability Models

during fault removal. The software failure rate, or the hazard function, at any time, is assumed to be proportional to the current fault content of the system. Thus, the random variable T_i, which denotes the time between the $(i-1)^{th}$ and i^{th} failure, has a density function given by

$$p(t_i|\lambda_i)) = \lambda_i e^{-\lambda_i t_i} \tag{1}$$

where λ_i is the transition rate, given by

$$\lambda_i = \phi[N-(i-1)] \tag{2}$$

where ϕ is a proportionality constant. The Markov chain for the Jelinski-Moranda model is as shown in the Figure 2.

The variation of the above model proposed by Moranda[16] is suited to model situations where the faults are not removed until a fatal one occurs. After the occurrence of a fatal fault, the accumulated group of faults is removed. This is the geometric de-eutrophication model and assumes the hazard function after restart to be a fraction of the rate which attained when the system crashed. The random variable T_i, has a exponential distribution as given by Equation (1). However, the transition rate, λ_i, for this model during the testing interval t_i is given by

$$\lambda_i = Dk^{i-1} \tag{3}$$

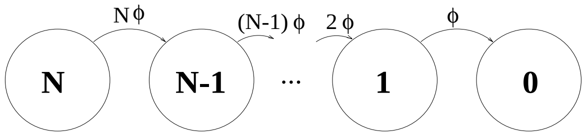

Figure 2 : Markov Chain - JM Model

Figure 2: Markov Chain - JM Model

where D is the fault detection rate during the first interval and k is a constant $(0 \leq k \leq 1)$.

A number of Bayesian extensions have been proposed to the basic JM model[13, 15]. The model proposed by Singpurwalla[15] assumes N to be unknown and is Poisson(λ) *a priori*, where λ is known. The distribution of the unknown number of defects is generalized further from Poisson(λ) by assuming the λ itself to be a random quantity with a Beta prior distribution[13]. Both the models assume ϕ to be a fixed quantity.

The JM model has a poor predictive capability in many cases. The Bayesian extension to the JM model by Littlewood[1] was proposed to improve the parameter estimation of the model and hence its predictive capability. This extension requires a reparametrization of the original JM model. The new parameters are (λ, ϕ), where $\lambda = N\phi$, and can be considered as the initial rate of occurrence of failures, and ϕ, the improvement resulting from a fix. The proposed model assumes uniform "ignorance prior", as the *a priori* distribution for (λ, ϕ).The Bayesian models can be viewed as a homogeneous Markov chain in a random environment.

Goel-Okumoto Imperfect Debugging Model[9]

The models discussed so far assume that the fault removal process is perfect. However, in practice this is not always the case. The Goel-Okumoto imperfect debugging model, which is an extension of the JM model, attempts to overcome this limitation. In this model, the transition rates are determined by the number of residual faults and also by the probability of imperfect debugging.

Littlewood Model[1]

In the JM model all the repairs have the same effect on software reliability. The Littlewood model is based on similar assumptions to those of the JM model except that different faults contribute differently to the unreliability of the software. Since the contribution of each fault to the unreliability is different, the sequence of failure intensities forms a stochastic process. The times between successive failures are exponentially distributed, as per Equation (1), and the transition rate is given by

$$\lambda_i = \phi_1 + \phi_2 + \ldots + \phi_{N-i+1} \tag{4}$$

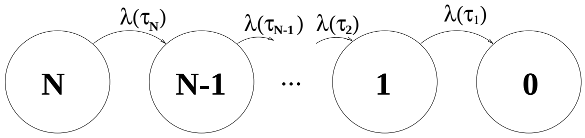

Figure 3 : Semi Markov Chain

Figure 3: Semi-Markov Chain

where the random variable ϕ_j represents the failure rate associated with fault j.

The initial rates $\phi_1, \ldots, \phi_N$ are assumed to be independent identically distributed (iid) gamma(α,β). After the program has executed for a total time of τ, the remaining rates are iid gamma(α,$\beta + \tau$). Since ϕ_j's are random variables, this model can be considered to be a Markov chain in a random environment. The Bayesian Extension to the above model proposed by Littlewood[1], also involves a reparameterization as in the case of JM model. The parameters considered are (λ, α, β) where $\lambda = N\alpha$. A complete Bayesian analysis of this model is computationally complex. Thus, the proposed model assumes β to be known, and independent gamma priors on the unknown (λ, α). The required quantities are then computed conditional on β. Finally an estimator of β is obtained using maximum likelihood technique and is substituted into the previously computed metrics.

4.2 Semi-Markov Models

For a definition of semi-Markov process the reader is urged to see [23]. The number of faults remaining in the software in this case is modeled using a semi-Markov process. As in the case of homogeneous Markov models, the number of faults remaining in the software product forms the state space of the Markov chain. However, the transition rates from each state depend not only on the number of remaining faults but also on the time spent in the current state. A representative model belonging to this class is discussed below. Figure 3 shows a semi-Markov chain, where $\lambda(\tau_i)$ indicates that the transition rate depends on τ_i, where τ_i is the sojourn time in state i.

Schick and Wolverton Model[24]

This model is based on the same assumptions as the Jelinski-Moranda model. The transition rate, λ_i, during the test interval t_i, is assumed to be proportional to the current fault content of the system, and the time elapsed since the last failure and is given by

$$\lambda_i = \phi[N - (i - 1)]t_i \tag{5}$$

where N and ϕ have the same interpretations as in the JM model. The hazard rate in this case is linear with time in each failure interval, returns to zero at the occurrence of a failure, and

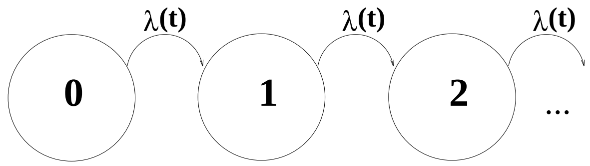

Figure 4 : Non-Homogeneous Markov Chain

Figure 4: Non-Homogeneous Markov Chain

increases linearly again, at a reduced slope. The decrease in slope is proportional to ϕ.

A variation of the above model[25] assumes the transition rate to be parabolic in test time and is given by

$$\lambda_i = \phi[N - (i-1)](-a{t_i}^2 + bt_i + c) \tag{6}$$

where a, b, c are constants and N and ϕ have the same interpretation as in the JM model. The transition rate in this case is the superposition of the transition rate of the JM model times a second term which states that the likelihood of a failure occurring increases rapidly as the test time accumulates within an interval. Note that at the beginning of a testing interval, i.e., $(t_i = 0)$, the transition rate is proportional to that of the JM model, with the constant of proportionality being c.

4.3 Non-Homogeneous Markov Models

For a definition of Non Homogeneous Markov chains the reader is urged to see [29]. This class of models is concerned with the number of faults detected in a given time and hence are referred to as "fault count models". The most popular subclass assumes that the number of failures in a given interval has a Poisson distribution whose parameter takes different forms for different models. The mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, of the software system satisfy the following relations

$$m(t) = \int_0^t \lambda(s)ds \tag{7}$$

$$\frac{dm(t)}{dt} = \lambda(t) \tag{8}$$

A non-homogeneous Markov chain for a general failure intensity function, $\lambda(t)$ is as shown in the Figure 4.

The NHPP models can be further classified as finite failures and infinite failures models.

4.3.1 Finite Failures NHPP Models

They assume that the expected number of failures observed during an infinite amount of time will be a finite number, a. Some of the key models in this class are described below.

Goel-Okumoto NHPP Model[9]

The Goel-Okumoto model is considered to be one of the most influential NHPP-based software reliability models. Its mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, are given by

$$m(t) = a(1 - e^{-gt}) \tag{9}$$

and

$$\lambda(t) = age^{-gt} \tag{10}$$

where g is the failure occurrence rate per fault.

The basic execution time model of Musa further assumes that the failure occurrence rate per fault is given by

$$g = fK \tag{11}$$

where f is the linear execution frequency of the program or the ratio of average instruction rate to program size in object instructions, and K is the fault exposure ratio.

The GO model assumes that the failure intensity of the software system decreases as testing progresses. However, initially the testing team is not familiar with the software, hence fault removal is slow, but after a certain time, the team gains sufficient experience and knowledge about the behavior of the product under test which leads to higher rates of fault removal until a time is reached when a large number of faults have been detected and removed, thus becoming increasingly more difficult to detect and remove new ones. Therefore, the failure rate increases initially and then decreases. A variation of the GO model, known as the Generalized GO model[9], was proposed to capture this increasing/decreasing failure rate. The mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, of the software system are given by

$$m(t) = a(1 - e^{-gt^{\gamma}}) \tag{12}$$

and

$$\lambda(t) = ag\gamma e^{-gt^{\gamma}} t^{\gamma-1} \tag{13}$$

where g and γ reflect the quality of testing.

Delayed S-Shaped NHPP Model[22]

The delayed S-shaped software reliability growth model was proposed to model the software fault removal phenomenon in which there is a time delay between the actual detection of the fault and its reporting. The test process in this case can be seen as consisting of two phases:

fault detection and fault isolation. The mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, of the software system in this case is given by

$$m(t) = a[1 - (1 + gt)e^{-gt}] \tag{14}$$

and

$$\lambda(t) = g^2 te^{-gt} \tag{15}$$

where g is the fault removal (failure detection and fault isolation) parameter.

Inflection S-Shaped NHPP Model[22]

The inflection S-shaped model was proposed to analyze the software failure detection process where the faults in a program are mutually dependent (some faults are not detectable before other faults are removed). The mean value function, $m(t)$, and the software failure intensity, $\lambda(t)$, are given by

$$m(t) = a\frac{1 - e^{-\phi^t}}{1 + \psi e^{-\phi t}} \tag{16}$$

and

$$\lambda(t) = a\frac{\phi e^{-\phi t}(1 + \psi)}{(1 + \psi e^{-\phi t})^2} \tag{17}$$

where ϕ is the failure detection rate in the sense of the JM model, and ψ is the inflection parameter.

Caruso[6] has proposed a methodology to integrate the available prior knowledge with inflection s-shaped model. This approach is based on using the prior knowledge to restrict a of the inflection s-shaped model between two values a_{max} and a_{min}.

C1 NHPP Model[18]

The failure occurrence rate per fault is assumed to be parabolic in this model. The mean value function, $m(t)$, is given by

$$m(t) = a\{1 - e^{-[(l/3)t^3 + (m/2)t^2 + nt]}\} \tag{18}$$

The corresponding software failure intensity, $\lambda(t)$ is given by

$$\lambda(t) = a(lt^2 + mt + n)e^{-[(l/3)t^3 + (m/2)t^2 + nt]} \tag{19}$$

where l, m and n are real-valued constants.

Pareto NHPP Model[18]

This model assumes the failure occurrence rate per fault to be Pareto. The mean value function, $m(t)$, and the failure intensity function, $\lambda(t)$, are given by

$$m(t) = a\left[1 - (1 + \frac{t}{\beta})^{-\alpha}\right] \qquad \alpha, \beta > 0 \tag{20}$$

and

$$\lambda(t) = \frac{a\alpha}{\beta}(1 + \frac{t}{\beta})^{-\alpha-1} \qquad \alpha, \beta > 0 \tag{21}$$

Littlewood NHPP Model[1]

The mean value function, $m(t)$, and the failure intensity function, $\lambda(t)$, of the Littlewood NHPP model are given by

$$m(t) = a\beta^{\alpha}[\frac{1}{\beta^{\alpha}} - \frac{1}{(\beta + t)^{\alpha}}] \qquad \alpha, \beta > 0 \tag{22}$$

and

$$\lambda(t) = \frac{a\alpha\beta^{\alpha}}{(\beta + t)^{\alpha+1}} \qquad \alpha, \beta > 0 \tag{23}$$

Hyperexponential NHPP Model[22]

The NHPP models described earlier assume that faults residing in a software product are of a single type. The hyperexponential software reliability model was proposed to analyze a failure detection process in module-structured software. The faults in these modules have different characteristics from the failure detection point of view. The hyperexponential growth model is a simple sum of the exponential growth models of clusters that have different failure intensities. The mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, is given by

$$m(t) = a(1 - \sum_{i=1}^{n} b_i e^{-g_i t}). \tag{24}$$

$$\lambda(t) = a \sum_{i=1}^{n} b_i g_i e^{-g_i t} \tag{25}$$

where,

$$b_i = \frac{a_i}{a} \tag{26}$$

Here n is the number of modules or clusters, a_i is the number of initial faults in cluster i, and g_i is the failure detection rate for cluster i.

Musa's Basic Execution Time (BET) model discussed earlier assumes uniform execution of instructions. The Extended Execution Time model (EET)[8] was proposed to take into account the non-uniform execution of instructions. The EET model partitions the software into cells of equal size, such that the execution of instructions in each cell is uniform. The failure occurrences in each cell can then be modeled using the BET model.

4.3.2 Infinite Failures Models

The mean value function of this class of models is unbounded, i.e., the expected number of failures experienced in infinite time is infinite. Some of the popular models in this class are described below:

Musa-Okumoto Logarithmic Poisson Execution Time Model[19]
This model assumes the number of failures experienced by time τ (τ is the execution time), is NHPP, with the mean value function, $m(\tau)$ given by

$$m(\tau) = \frac{1}{\theta}\ln(\lambda_0\theta\tau + 1) \tag{27}$$

where λ_0 denotes the initial failure intensity, and $\theta > 0$, the failure decay parameter.

The failure intensity is given by

$$\lambda(\tau) = \frac{\lambda_0}{(\lambda_0\theta\tau + 1)} \tag{28}$$

A Bayesian approach for predicting the number of failures in a software product using the logarithmic NHPP model has been proposed by Campodónico and Singpurwalla [5]. This model uses, in a formal manner, expert knowledge in the conduct of software testing.

Duane Model[7]
The Duane model was originally proposed for hardware reliability studies. Its mean value function, $m(t)$, and the failure intensity function, $\lambda(t)$, are given by

$$m(t) = at^b, \qquad b > 0 \tag{29}$$

and

$$\lambda(t) = abt^{b-1} \qquad b > 0 \tag{30}$$

This model usually overestimates the cumulative number of failures. The main criticism of the model is that its mean value function approaches infinity very rapidly.

Log-Power NHPP Model[31]
The mean value function, $m(t)$, and the failure intensity function, $\lambda(t)$, of the log-power model are given by

$$m(t) = a\ln^b(1+t) \qquad b > 0 \tag{31}$$

$$\lambda(t) = \frac{ab\ln^{b-1}(1+t)}{1+t} \qquad b > 0 \tag{32}$$

The mean value function of the log-power satisfies the following relation

$$\ln m(t) = \ln a + b \ln[\ln(1+t)] \tag{33}$$

Thus, if the model is suitable for the data, the plot of $m(t)$ vs. $(t+1)$, on a log-loglog will tend to be a straight line. This technique enables partial validation of the model prior to embarking upon the parameter estimation.

4.4 Other Models

In this section, we discuss some of the time-domain models which can neither be classified as homogeneous Markov nor non-homogeneous Markov models.

Littlewood-Verall (LV) Bayesian Model[1]
The Littlewood-Verall (LV) Bayesian Model assumes that the random variable T_i, which denotes the time between the $(i-1)^{th}$ and i^{th} failure, has a density function given by

$$p(t_i | \Lambda_i = \lambda_i) = \lambda_i e^{-\lambda_i t_i} \tag{34}$$

Further, the $\{\Lambda_i\}$ are treated as random variables with a gamma density function,

$$p(\lambda_i) = \frac{[\psi(i)]^\alpha {\lambda_i}^{\alpha-1} e^{-\psi(i)\lambda_i}}{\Gamma(\alpha)} \tag{35}$$

The function $\psi(i)$ determines the reliability growth. If $\psi(i)$ is an increasing function of i, the sequence $\{\Lambda_i\}$ forms a stochastically decreasing sequence. The user can choose a specific parametric family for $\psi(i)$ to meet his requirements.

Littlewood and Keiller (LK) Bayesian Model[1]
This model is similar to LV model, except that the reliability growth is introduced via the shape parameter of the gamma distribution for the failure intensities. The failure intensities have a gamma density function given by

$$p(\lambda_i) = \frac{\beta^{\psi(i)} {\lambda_i}^{\psi(i)-1} e^{-\beta\lambda_i}}{\Gamma(\psi(i))} \tag{36}$$

The reliability growth in this case occurs when $\psi(i)$ is a decreasing function of i. The choice of the parametric family for $\psi(i)$ is up to the user.

Random Coefficient Autoregressive Model[26]
This model assumes, X_i, the time to failure of the software after i^{th} modification is made to

it has a lognormal distribution. The model is based on the power law which relates lifetimes from one stage to the next. Analytically, the model is given by

$$X_i = {X_{i-1}}^{\theta_i} \delta_i \tag{37}$$

where θ_i is a coefficient and δ_i accounts for deviations in the specification of power law as a model for reliability growth. The values of θ_i dictates the growth or decay in reliability. The power law can also be expressed as

$$Y_i = \theta_i Y_{i-1} + \epsilon_i \tag{38}$$

where $Y_i = \log X_i$. Y_i and ϵ_i are both normally distributed. The model further assumes *a priori* normal distribution for θ_i.

Thus, the sequence $\{Y_i\}$ is described by a first order autoregressive process with a random coefficient.

5 Data Domain Models

We now briefly discuss some of the models that fall under the broad category of "data-domain" models.

5.1 Fault Seeding Models

In this class of models, a known number of faults are planted in the program. The total number of seeded and indigenous faults uncovered as a result of testing are then counted. Using combinatorics and the maximum likelihood technique, the number of indigenous faults are estimated, and the reliability of the software is then computed.

Mill's Hypergeometric Model[17]

This is the most popular and the most basic fault seeding model. It is based on the standard capture-recapture (C-R) sampling technique. In this model, a known number faults, N_1, are seeded in a software product, which originally has N indigenous faults; N is assumed to be unknown and it is estimated from the number of indigenous and seeded faults observed during the test using a hypergeometric distribution. The probability that exactly k out of r detected faults are seeded faults is given by

$$q_k(N) = \frac{\binom{N_1}{r}\binom{N-N_1}{r-k}}{\binom{N}{r}} \tag{39}$$

The maximum likelihood estimate of N, denoted as $\hat{N}$, is given by

$$\hat{N} = \left[\frac{N_1 r}{k}\right] \tag{40}$$

The procedure adopted in this model is analogous to the one used for estimating the number of fish in a pond or for estimating wildlife.

In Equation (39), Mills assumes that the probability of selecting a sample of size r, is given by

$$p = \binom{N}{r} \tag{41}$$

Since the total number of faults (seeded and indigenous) in the software is $N + N_1$, the above probability is incorrect, as pointed out by Basin[2] and the actual probability is given by

$$p = \binom{N+N_1}{r} \tag{42}$$

The MLE estimate of N, is thus given by

$$\hat{N} = \left[\frac{N_1(r-k)}{k}\right] \tag{43}$$

Basin[2] has suggested a two-stage testing procedure where one programmer searches for, detects, and records N_1 faults out of a total of N unknown indigenous faults. A second programmer is assigned the task of independently testing the program, and records r out of N possible faults. The two lists of faults are then compared, and Equation (39) represents the probability that k faults in the second programmer's list are included in the first programmer's list.

5.2 Input Domain Based Models

In the case of the input domain models, the basic method involves the generation of test cases from an input distribution which represents the operational usage of the program. Since it is difficult to obtain this distribution, the input domain is usually partitioned into a set of equivalence classes. Each of the equivalence classes represents a program path. An estimate of reliability is the ratio of the number of input test cases that result in successful execution to the total number of sampled input cases.

Nelson Input Domain Model[27]

The reliability is estimated by running the software for a set of n inputs. The inputs are chosen randomly from a set $\{E_i : i = 1, 2, \ldots, N\}$. Each E_i is a set of data values needed to make a run. The random sampling of n inputs is done according to a probability vector P_i, where P_i is the probability that E_i is sampled. The probability vector $\{P_i : i = 1, 2, \ldots, N\}$ defines the operational profile or the user input distribution. If the number of failures is f, then the estimate of reliability is given by

$$\hat{R} = 1 - \frac{f}{n} = \frac{n-f}{n} \tag{44}$$

Brown and Lipow Input Domain Model[4]
The Nelson model explicitly incorporates the usage distribution or the test case distribution. In the model proposed by Brown and Lipow, it is assumed that the accomplished testing is representative of the expected user distribution. In this model, the entire input domain is partitioned into subdomains. Each E_i from the input domain $\{E_i : i = 1, 2, \ldots, N\}$ represents a specific subdomain. The estimated reliability in this case is given by

$$\hat{R} = 1 - \sum_{i=1}^{N} \left(\frac{f_j}{n_j} \right) P(E_j) \tag{45}$$

Here n_j is the number of test cases sampled from subdomain E_j, f_j is the number of test cases which resulted in an abnormal execution out of n_j test cases, and $P(E_j)$ is the probability that inputs in domain E_j are used in an actual operational environment.

Ramamoorthy and Bastani Input Domain Model[3]
This input domain model is concerned with the reliability of critical, real-time process control systems. The metric here is the confidence in the reliability estimate rather than the reliability estimate itself. The model provides a conditional probability that the program is correct for all possible inputs of interest given that it results in successful execution for a specified subset of the inputs.

6 Comparison of Time Domain and Data Domain Models

The time domain models emphasize the underlying failure process while the data domain models focus on the fault content of the software product under consideration. These two approaches provide completely different perspectives on software reliability. The time-domain models provide close form analytical expressions and are more economical to apply than their data-domain counterparts. Usually data-domain models require very lengthy testing with a representative and statistically meaningful set of inputs for their validation. Time-domain models are also well-suited for tightly scheduled projects or projects with simple structured software. In the case of complex structured software such as systems programs and communication programs, the time domain models tend to underestimate the number of remaining errors[21]. Analyzing both the failure content and failure process could increase the accuracy of reliability estimation, and an integrated approach based on these two fundamentally different philosophies may be beneficial[28].

7 Concluding Remarks

In this paper, we have provided a broad classification of the previously reported software reliability models, and discussed some of the representative models belonging to each of the categories. The list of models addressed here is by no means exhaustive. An overwhelming number of models has been proposed to address the issue of software reliability assessment, but no single model can be recommended unreservedly to potential users under all circumstances. One must recognize that the faults are not uniformly distributed nor of same severity, and that fault repair does not always result in a better product since there is an opportunity for introducing new faults during the repair process. There are currently known research efforts that are dealing with these issues[11]

References

[1] A.A. Abdel-Ghally, P.Y. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 9, September 1986.

[2] S.L. Basin, "Estimation of Software Error Rates via Capture- Recapture Sampling," Science Applications, Inc., Palo Alto, CA, Sept. 1973.

[3] C.V. Ramamoorthy and F.B. Bastani, "Software Reliability-Status and Perspectives," *IEEE Trans. on Software Engineering*, vol. SE-8, No. 4, July 1982.

[4] J.R. Brown and M. Lipow, "Testing for Software Reliability," *Proc Int'l Conf. Reliable Software*, pp. 518-527, Los Angeles, CA, April 1975.

[5] S. Campodónico and N.D. Singpurwalla, "A Bayesian Analysis of the Logarithmic-Poisson Execution Time Model Based on Expert Opinion and Failure Data," *IEEE Trans. on Software Engineering*, vol. 20, No. 9, September 1994.

[6] J.M. Caruso and D.W. Desormeau, "Integrating Prior Knowledge with a Software Reliability Growth Model," *Proc. of Intl. Conf. on Software Engineering*, 1991.

[7] J.T. Duane, "Learning Curve Approach to Reliability Monitoring," *IEEE Trans. Aerosp.*, vol. AS-2, pp. 563-566, 1964.

[8] W.W. Everett, "An "Extended Execution Time" Software Reliability Model," *Proc. of 3rd Intl. Symposium on Software Reliability Engineering*, 1992.

[9] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 12, December 1985.

[10] A.L. Goel and K. Okumoto, "An Analysis of Recurrent Software Failures in a Real-Time Control System," in *Proc. ACM Annu. Tech. Conf.*, ACM, Washington DC, 1978, pp. 496-500.

[11] S. Gokhale, T. Philip, P.N. Marinos and K.S. Trivedi, "A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair," Submitted to the *IEEE Intl. Computer Performance and Dependability Symposium.* Urbana-Champaign, IL, Sept. 1996.

[12] Z. Jelinski and P.B. Moranda, "Software Reliability Research," *Statistical Computer Performance Evaluation* (W. Freiberger, *Ed*); pp. 465-484, Academic Press, New York, 1972.

[13] W.S. Jewell, "Bayesian Extensions to a Basic Model of Software Reliability," *IEEE Trans. on Software Engineering*, vol. SE-11, No. 12, December 1985.

[14] P.A. Keiller, B. Littlewood, D.R. Miller and A. Sofer, "Comparison of Software Reliability Predictions," in *13th Int. Symp. Fault-Tolerant Comput.*, 1983, pp.128-134.

[15] R.J. Meinhold and N.D. Singpurwalla, "Bayesian Analysis of a Commonly Used Model for Describing Software Failures," *The Statistician*, vol. 32, pp. 168-173, 1983.

[16] P.B. Moranda, "Prediction of Software Reliability during Debugging," in *Proc. Annu Reliability and Maintainability Symp.*, Washington, DC, Jan. 1975, pp. 327-332.

[17] H.D. Mills, "On the Statistical Validation of Computer Programs," IBM Federal Syst. Div., Gaithersburg, MD, Rep. 72-6015, 1972.

[18] J.D. Musa and K. Okumoto, "A Comparison of Time Domains for Software Reliability Models," *Journal of Systems and Software*, pp. 277-287, 1984.

[19] J.D. Musa, "A Theory of Software Reliability and its Application," *IEEE Trans. on Software Engineering*, vol. SE-1, No. 3, September 1975.

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

[21] Y. Nakagawa and S. Hanata, "An Error Complexity Model for Software Reliability Measurement," *Proc. Intl. Conf. on Software Engineering*, 1989.

[22] M. Ohba, "Software Reliability Analysis Models," *IBM J. Res Develop*, Vol. 28, No. 4, July 1984.

[23] R.A. Sahner, K.S. Trivedi and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*, Kluwer Academic Publishers, Boston, 1995.

[24] G.J. Schick and R.W. Wolverton, "Assessment of Software Reliability," presented at the 11th Annu. Meeting German Oper. Res. Soc., DGOR, Hamburg, Germany; also in *Proc. Oper. Res.,* Physica-Verlag, Wirzburg-Wien, 1973, pp. 395-422.

[25] G.J. Schick and R.W. Wolverton, "An Analysis of Computing Software Reliability Models," *IEEE Trans. Software Engineering*, vol. SE-4, pp. 104-120, July 1978.

[26] N.D. Singpurwalla and R. Soyer, "Assessing (Software) Reliability Growth Using a Random Coefficient Autoregressive Process and its Ramifactions," *IEEE Trans. on Software Engineering*, vol. SE-11, No. 12, Dec. 1985.

[27] T.A. Thayer, M. Lipow and E.C. Nelson, "Software Reliability Study," Rep. RADC-TR-76-238, Aug. 1976.

[28] J. Tian, "Integrating Time Domain and Input Domain Analyses of Software Reliability Using Tree-Based Models," *IEEE Trans. on Software Engineering*, vol. 21, No. 12, December 1995.

[29] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[30] S. Yamada, M. Ohba and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Trans. on Reliability*, Vol. R-32, No.5, December 1983.

[31] M. Zhao and M. Xie, "On the Log-Power NHPP Software Reliability Model," *Proc. of 3rd Intl. Symposium on Software Reliability Engineering*, 1992.