The discrete time hyperexponential model for software reliability growth evaluation

Mohamed Kaâniche, Karama Kanoun

The Discrete Time Hyperexponential Model for Software Reliability Growth Evaluation

M. Kaâniche K. Kanoun

LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse (France)

Abstract

This paper is devoted to the definition of a reliability growth model—referred to as the discrete time hyperexponential model. This model is aimed at modeling software reliability with respect to the number of executions performed. It is well-suited to some kinds of systems such as transaction processing systems, single mission systems, etc… for which discrete data are collected; in addition, it facilitates the modeling of the impact on dependability measures of some software environment characteristics, such as the input probability distribution.

1. Introduction

The rapid growth in size, complexity and cost of computing systems and their applications, has led to a large amount of research devoted to software reliability modeling and evaluation. An important number of software reliability growth models has been defined to follow up the software behavior during its validation and operation phases and estimate the software ability to deliver a service that complies with the specification (see for example [16], [17] [19], [30]). The most well known proposed models are continuous time models based on the characterization of the software failure process with respect to the system execution time or calendar time. Software reliability is assessed through the evaluation of measures such as mean time to failure, failure rate, failure intensity, etc….

However, only few models, termed discrete time reliability models, allowing software reliability to be assessed with respect to the number of executions or runs performed rather than time have been reported in the literature [3], [23], [31]. Nevertheless, this kind of models deserves particular attention. In fact, discrete time representation of the software failure process is well suited for systems for which it is more significant to consider the number of executions performed rather than time; for instance transaction processing systems and single mission systems. It is also appropriate for the follow up of the software behavior during its validation phase and for the assessment of its reliability when discrete data are collected (number of successful executions between failures, for example). These data, are recorded, for instance, during statistical testing of the software with test inputs selected from an input distribution representative of the user profile.

In this paper, a reliability growth model referred to as the discrete time hyperexponential model, is proposed for the modeling and evaluation of software reliability with respect to the number of executions performed. Even though this model is based on similar assumptions to those considered for the establishment of the continuous time hyperexponential model presented in [13], [15] and applied to real systems in [11], [12], its aims are quite different. Particularly, it is better suited to take into account some characteristics of the software execution profile, such as the input probability distribution, in reliability evaluation.

This paper is composed of five Sections. Basic discrete time reliability measures are first presented in Section 2. In order to position our model with respect to existing discrete time software reliability models, an overview of these models is given in Section 3. In Section 4, the discrete time hyperexponential model is defined, the main related reliability measures are derived, then the model is applied to field data. In Section 5, the hyperexponential model is used to evaluate software reliability growth by taking into account explicitly the software input probability distribution characterizing the execution profile in which the software is run.

2. Basic discrete time reliability measures

To our knowledge, only few results have been reported about the discrete time reliability theory [10], [25]. We present in this section the definitions of and relationships between the discrete time reliability function and the associated quantities.

Let N be the random variable representing the number of executions until failure occurrence. N can be characterized by the following quantities:

- *probability mass function*: $f(n) = Pr\{N=n\}$,
- *probability distribution function*: $F(n) = Pr\{N \leq n\}$, $n > 0$ and $F(0) = 0$,
- *reliability function*: $R(n) = Pr\{N>n\} = 1 - F(n)$, $n > 0$ and $R(0) = 1$.
- *discrete time failure rate*: $p(n) = Pr\{N=n \mid N \geq n\}$.

The relationship between R(n), f(n) and p(n) is summarised in Fig. 1: note that the specification of one of these quantities allows the others to be derived.

	R(n)	f(n)	p(n)
R(n)	*	$\sum_{j=n+1}^{\infty} f(j)$	$\prod_{j=1}^{n}[1- p(j)]$
f(n)	R(n-1)-R(n)	*	$p(n)\prod_{j=1}^{n-1}[1- p(j)]$
p(n)	$\frac{R(n-1)-R(n)}{R(n-1)}$	$\frac{f(n)}{\sum_{j=n}^{\infty} f(j)}$	*

Figure 1: Relationships between discrete time reliability measures

In addition to the previous measures, we can evaluate the mean number of executions until failure, noted MTTF, which is the expected value of the random variable N, defined by:

$$MTTF = E[N] = \sum_{n=1}^{\infty} n\, f(n) \quad (1)$$

Given that: $f(n) = R(n-1) - R(n)$, relation (1) can be rewritten as follows:

$$MTTF = \sum_{n=1}^{\infty} n\, R(n-1) - \sum_{n=1}^{\infty} n\, R(n) \quad (2)$$

Relation (2) yields:

$$MTTF = \sum_{n=0}^{\infty} R(n) \quad (3)$$

The measures defined above concern the occurrence of one failure, the first failure for instance, since it is assumed that the time of the beginning of observation is zero. For failure i, the previous measures have to be conditioned on the time of occurrence of failure i-1, let m_{i-1} be this time. For example, the probability mass function becomes: $f(n_i \mid m_{i-1})$.

3. Discrete time software reliability models: A survey

Discrete time software reliability modeling has given rise to a few models that can be classified into two categories:

- *stable reliability* models which characterize the software failure process when no fault removal is performed; these models assume that the software probability of failure at execution remains constant,
- *reliability growth* models which are prediction models taking into account the stochastic decrease of software probability of failure at execution originating from progressive removal of design faults.

3.1. Stable reliability models

Models assuming stable reliability behavior of the software are aimed at evaluating estimators for software reliability when the input data are sampled statistically according to a probability distribution that is expected to be representative of the operational software execution profile. The most popular discrete time stable reliability model was proposed by Nelson [20], [21]. For this model, an execution or a run of the software corresponds to the selection of a point from the input domain. If n executions are performed during which n_f failures are observed, then an unbiased estimator for reliability after n runs is given by: $\hat{R} = 1 - \frac{n_f}{n}$. As no fault removal is supposed to be performed during the software exposure period, and input data are assumed to be selected independently and randomly from the input domain, the software probability of failure at execution is assumed to be constant and the number of executions up to failure follows a geometric distribution.

Even though the Nelson model is based on theoretical foundations that are sound, it suffers from practical drawbacks that are detailed in [24], for example: i) a large number of executions is needed in order to obtain a high confidence in the reliability estimate, and ii) the probability input distribution is not taken into account explicitly in reliability estimation. In order to overcome some of these

drawbacks, extensions to and generalizations of the Nelson model have been proposed; see for example [1], [24], [29][1].

The previous models require a large sample of failure data to be collected in order to estimate software reliability. When either zero failures or only a few failures are observed during software testing, these models are no longer suitable for reliability estimation. For these situations, other approaches based on software statistical testing have been proposed in order to quantify a lower bound for software reliability for a given confidence level; for instance, [2],[4], [22].

3.2. Reliability growth models

As far as we know, only a few models encompassed in this category have been reported [3], [7], [23], [27], [31]. A brief description of these models is given in the sequel.

In [23], an input domain based stochastic reliability growth model is defined. The software failure process is characterized by the set of parameters p_j which correspond to the software discrete time failure rate after j modifications being performed. Conditionally on p_j the number of executions until failure occurrence is supposed to be geometrically distributed. In order to take account of the reliability growth phenomenon as well as the uncertainty about the consequence of software changes on its behavior, p_j is modeled by a random walk stochastic process that satisfies the following requirements:

$$\Delta_j = p_{j-1} - p_j \underset{st}{\leq} \Delta_{j-1} \text{ and } p_j \underset{st}{\leq} p_{j-1}$$

Based on these assumptions, software reliability is assessed through the evaluation of the mean time to failure given that j modifications have occurred.

The approach considered in [3] is quite different since a deterministic expression for the discrete time failure rate is assumed. A relationship between the software discrete time failure rate and the distribution of faults in software paths executed is established when either a uniform or a nonuniform path selection strategy is assumed.

For the previous mentioned models, the software failure process is characterized by a piecewise stochastic discrete time process such that the variation of the failure rate takes place at software modification. The discrete time failure rate is stochastically decreasing with the number of executions performed and reaches asymptotically a zero limiting value.

Parallel to continuous time reliability growth models based on non homogeneous Poisson processes, analogous discrete time reliability growth models have also been developed [7], [31]. For these models, no explicit relationship between the number of software modifications and discrete time failure rate variation is assumed. In [31], the software failure process is characterized by a discrete counting process representing the cumulative number of failures observed out of n executions such that the associated mean value function has an exponential growth curve. In [7][2], a logarithmic growth relationship between the number of executions performed and the number of failures observed is assumed.

Additionally to the previous models, one may also mention the model presented in [27] that is devoted to the prediction of the cumulative number of software failures with respect to the number of test instances applied during software testing. During a test instance, many software executions may be performed and more than one fault can be activated. For this model, a distinction is made between manifestation and detection of faults at the application of a test instance; a hypergeometric distribution is considered for the estimation of the number of initial faults in the software and attention is focussed on the evaluation of the cumulative number of failures in order to follow up the software behavior during testing.

An assumption shared by all reliability growth models mentioned above is that the software discrete time failure rate reaches asymptotically a zero limiting value. This supposes that all design faults can be removed from the software. However, for large software systems this assumption may not be satisfied. For these systems, the asymptotic software behavior is better represented by a stable reliability behavior characterized by a constant asymptotic discrete time failure rate, enabling software reliability to be evaluated when either: a) modifications are no longer performed, or when b) elimination of faults does not significantly affect the software failure process.

The discrete time hyperexponential model proposed in this paper enables this asymptotic stable reliability behavior to be taken into account. Additionally, it allows the evaluation of reliability growth of software multicomponent systems from the reliability growth of their components. The main features of this model are introduced in the following sections. Due to space limitations, only single component systems will be considered; multicomponent systems are dealt with in [8], [9].

---

[1] These models as well as the Nelson model are usually referred to as input-domain-based models.

[2] This model was defined in order to estimate reliability measures of single mission systems, not necessarily software systems.

4. The Hyperexponential model

The key objective of the discrete time hyperexponential model is to represent the reliability growth phenomenon and to evaluate measures allowing software reliability to be assessed with respect to the number of executions performed.

A software program can be seen as the mapping of its input domain I into its output space O. An execution of the system consists in selecting a sequence of input points not necessarily contiguous from the input domain. Due to the presence of internal design faults, the selection of some inputs from I, those sampled from the failure domain, may lead to erroneous outputs which differ from the specified ones, leading to software failure. As the failure domain is intercepted randomly during software execution, there is thus a non zero probability to fail at each input selection. The software execution process may then be seen as a series of independent Bernouilli trials where a trial corresponds to one software execution. Let:

- Z_n be a binary random variable characterizing execution n of the software:

  $\begin{cases} Z_n = 1 \text{ if execution n fails} \\ Z_n = 0 \text{ otherwise} \end{cases}$

- P(n) be the probability of failure at execution n; i.e., the probability of occurrence of the event $\{Z_n = 1\}$.

Given the assumed stochastic independence between the random variables Z_n, by analogy with continuous time non homogeneous Poisson process models, P(n) is equivalent to the software failure intensity h(t) which is the probability of failure in the interval [t, t+dt].

The discrete time hyperexponential model is aimed at modeling a continuously decreasing probability of failure at execution characterized by a constant limiting value. It is defined by the probability of failure at execution given by:

$$P(n) = \frac{\theta p_{sup}(1-p_{sup})^{n-1} + \bar{\theta} p_{inf}(1-p_{inf})^{n-1}}{\theta(1-p_{sup})^{n-1} + \bar{\theta}(1-p_{inf})^{n-1}} \qquad (4)$$

with $0 \le \theta \le 1, \bar{\theta} = 1-\theta$ and $p_{inf} \le p_{sup}$.

4.1. Model properties

As indicated in Fig. 2, the probability of failure at execution P(n) given by relation (4) is non increasing with the number of executions performed, when $0<\theta<1$, from $P(1) = \theta p_{sup} + \bar{\theta} p_{inf}$ to $P(\infty) = p_{inf}$. In fact, as the monotonicity of P(n) is indicated by the sign of the quantity $\Delta P(n+1) = P(n+1) - P(n)$, we have shown that [8]:

$\Delta P(n+1) =$ - **Erreur !** which is always negative for $n \ge 1$.

The rate of decrease of P(n) can be adjusted via the values of the parameters θ, p_{sup}, p_{inf}. The curvature of P(n) may change according to the value of parameter θ.

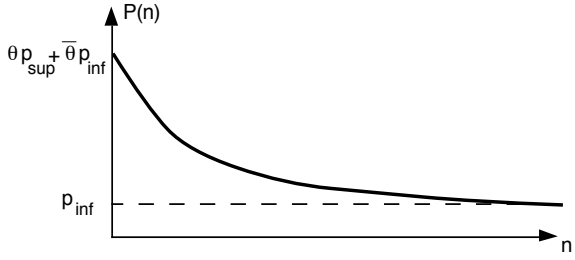


Figure 2: Typical probability of failure at execution for the discrete time hyperexponential model

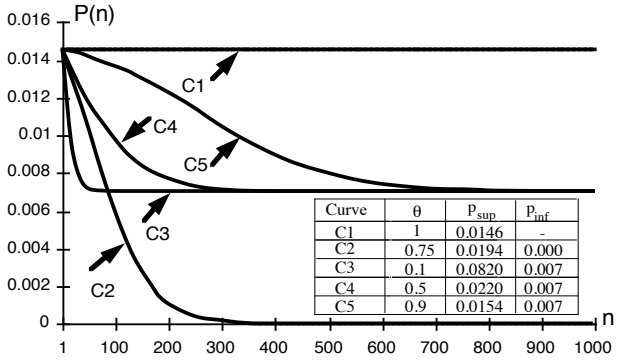
In fact, the curvature of P(n) is given by the sign of $\Delta P(n+1) - \Delta P(n)$. It can be seen that:

- if $\theta \le \frac{1}{2}$, the curvature of P(n) is always positive,
- if $\theta > \frac{1}{2}$, the curvature of P(n) is first negative, then positive for $n \ge n_0$:

$$n_0 = 1 + \frac{1}{\text{Ln}\left(\frac{1-p_{inf}}{1-p_{sup}}\right)} \text{Ln}\left(\frac{\theta}{\bar{\theta}}\right)$$

Figure 3 displays some typical examples of curves of probability of failure at execution represented by the discrete time hyperexponential model:

- C1 characterizes stable reliability behavior,
- C2 characterizes reliability growth tending to an asymptotic behavior with $P(\infty) = 0$,
- C3, C4, C5 respectively characterize fast, progressive and slow reliability growth with non zero asymptotic probability of failure at execution.



Curve	θ	p_{sup}	p_{inf}
C1	1	0.0146	-
C2	0.75	0.0194	0.000
C3	0.1	0.0820	0.007
C4	0.5	0.0220	0.007
C5	0.9	0.0154	0.007

Figure 3: Probability of failure at execution for the discrete time hyperexponential model

Note that the discrete time hyperexponential model admits as special cases:

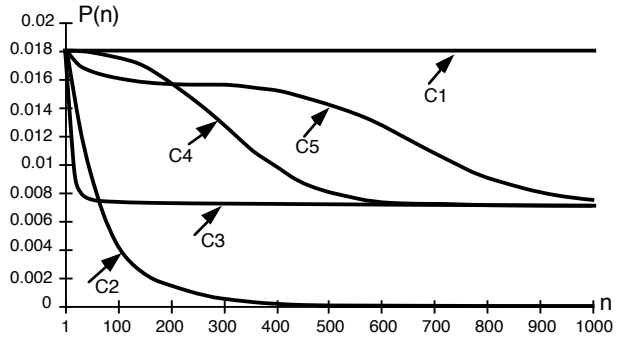
- the stable reliability situation, with constant probability of failure at execution:
  a) $p_{sup} = p_{inf}$, or b) $\theta = 0$ or $\theta = 1$;
- a probability of failure at execution tending asymptotically toward zero: $p_{inf} = 0$.

Hence, software reliability behaviors represented by existing software discrete time reliability models can be taken into account by the hyperexponential model as special cases.

Given that the probability of failure at execution relative to the hyperexponential model is based on a two stage discrete time Cox law; the proposed model can be generalized by introducing more stages in relation (4). The general expression for P(n) would be:

$$P(n) = \frac{\sum_{i=1}^{k} \theta_i p_i (1-p_i)^{n-1}}{\sum_{i=1}^{k} \theta_i (1-p_i)^{n-1}} \text{ with } 0 \le \theta_i \le 1 \text{ and } \sum_{i=1}^{k} \theta_i = 1 \qquad (5)$$

Figure 4 displays some examples of curves of probability of failure at execution relative to the generalized discrete time hyperexponential model when k=3. Comparison of Fig. 4 with Fig. 3, shows that the introduction of additional parameters into the expression of the probability of failure at execution enables better fitting of the reliability of software systems for which the probability of failure at execution varies slowly with respect to the number of executions performed (curve C5): two curvature changes of P(n) before reaching the asymptotic behavior can be taken into account. Note that, for the generalized discrete time hyperexponential model, the number of curvature changes increases with k. Hence, generalization of the model will enable better fitting to real situations, at the expense of added complexity estimation.



Curve	θ_1	θ_2	p_1	p_2	p_3
C1	1.00	0.000	0.018	0.010	0.007
C2	0.45	0.300	0.034	0.010	0.000
C3	0.10	0.100	0.115	0.010	0.007
C4	0.97	0.001	0.018	0.007	0.007
C5	0.10	0.897	0.038	0.016	0.007

Figure 4: Probability of failure at execution for the generalized hyperexponential model (k=3)

4.2. Derivation of related reliability measures

4.2.1 Reliability function

Let:

- N_i be the random variable representing the number of software executions performed between the occurrence of failures (i-1) and i, and n_i a realization of N_i,
- M_i be the random variable representing the cumulative number of software executions performed until the occurrence of failure i, and m_i a realization of M_i. m_i is the time of occurrence of failure i.
  $(M_i = \sum_{j=1}^{i} N_j)$.

The conditional reliability of N_i on the last failure time $M_{i-1}=m_{i-1}$ is defined by:

$$R(n_i \mid m_{i-1}) = Pr\{N_i \ge n_i \mid M_{i-1}=m_{i-1}\}$$

$$= \prod_{j=m_{i-1}+1}^{m_{i-1}+n_i} [1- P(j)] \qquad (6)$$

It follows that:

$$R(n_i \mid m_{i-1}) = \frac{\theta(1-p_{sup})^{m_{i-1}+n_i} + \bar{\theta}(1-p_{inf})^{m_{i-1}+n_i}}{\theta(1-p_{sup})^{m_{i-1}} + \bar{\theta}(1-p_{inf})^{m_{i-1}}} \qquad (7)$$

4.2.2. Probability mass function and failure rate

Using the definition of $f(n_i \mid m_{i-1})$ the probability mass function associated with N_i and conditional on the last failure time $M_{i-1}=m_{i-1}$ given in Section 2, we have shown in [8] that:

$$f(n_i \mid m_{i-1}) = P(m_{i-1}+n_i)\ \ R(n_i - 1 \mid m_{i-1}) \tag{8}$$

where $P(m_{i-1}+n_i)$ is the software probability of failure at execution $m_{i-1}+n_i$.

Using the relationships given in Section 2 and relation (8), it follows that the software discrete time failure rate after the occurrence of failure i-1, denoted $p(n_i \mid m_{i-1})$, is given by:

$$p(n_i \mid m_{i-1}) = P(m_{i-1}+n_i) \tag{9}$$

Hence, the discrete time failure rate cannot be distinguished from the non conditional probability of failure at execution; only the origin of time changes. Therefore we obtain a result analogous to non homogeneous Poisson processes (NHPP) for which the failure intensity and the failure rate have the same expression [18].

4.2.3. Mean cumulative number of failures

In addition to the basic reliability measures derived above, we can evaluate the cumulative number of failures with respect to the number of executions performed. This measure can be used as an appropriate index for the management of the validation and the maintenance effort needed to remove software design faults that are not revealed yet.

Let Y_n be the random variable representing the cumulative number of software failures experienced after n executions, and H(n) the s-expectation of Y_n. Considering the Bernouilli random variables Z_n, we have $Y_n = Z_1 + \ldots + Z_n$. It is shown that [6]:

$$H(n) = E[Y_n] = \sum_{i=1}^{n} E[Z_i] = \sum_{i=1}^{n} P(i) \tag{10}$$

$$H(n) = \textbf{Erreur !} \tag{11}$$

Generally, we have $p_{inf}, p_{sup} << 1$, which leads to the simplified expression:

$$H(n) \approx -Ln \left\{ \theta(1-p_{sup})^n + \bar{\theta}\,(1-p_{inf})^n \right\} \tag{12}$$

Proof:

When p_{inf} and $p_{sup} << 1$,

$1-P(i) \approx \exp(-P(i)) + o(p_{inf}, p_{sup})$ for $i = 1, 2\ldots$

$$\text{So, } \prod_{i=1}^{n} (1-P(i)) \approx \prod_{i=1}^{n} \exp(-P(i)) + o(p_{inf}, p_{sup})$$

$$\prod_{i=1}^{n} (1-P(i)) \approx \exp\left(- \sum_{i=1}^{n} P(i) \right) + o(p_{inf}, p_{sup})$$

$$\approx \exp(-H(n)) + o(p_{inf}, p_{sup}) \tag{13}$$

Given relation (4), it can be shown that:

$$\prod_{i=1}^{n} (1-P(i)) = \theta(1-p_{sup})^n + \bar{\theta}\,(1-p_{inf})^n \tag{14}$$

Substituting (14) into (13) yields to :

$$\exp(-H(n)) \approx \theta(1-p_{sup})^n + \bar{\theta}\,(1-p_{inf})^n \tag{15}$$

Taking the logarithm of (15) leads then to relation (12).

4.2.4. Distribution of cumulative number of failures

In addition to the evaluation of the expected value of Y_n, it is useful, particularly for the estimation of model parameters, to evaluate its associated probability distribution function. The probability distribution function of Y_n is difficult to obtain in an explicit form because of the non stationarity of P(i). Nevertheless, based on the assumption p_{inf} and $p_{sup} << 1$, it is proved [26], that the probability distribution function of Y_n can be approximated by a Poisson distribution characterized by the mean function $H(n) = \sum_{i=1}^{n} P(i)$. Therefore:

$$\Pr\{Y_n = i\} = \frac{[H(n)]^i}{i!} \exp\{-H(n)\} \tag{16}$$

Suppose that y_k failures have been observed during [0,k] executions, then it can be proved easily that the conditional distribution of Y_n given that $Y_k = y_k$ for $n > k$ is the distribution of the number of failures during [k+1,n], i.e.,

$$\Pr\{Y_n = y_n \mid Y_k = y_k\} = \Pr\{Y_n - Y_k = y_n - y_k\}$$

$$= \frac{[H(n) - H(k)]^{y_n - y_k}}{(y_n - y_k)!} \exp\{-[H(n) - H(k)]\} \quad (17)$$

4.3. Parameter estimation

Application of the discrete time hyperexponential model to real-life software systems necessitates the collection of failure data and the estimation of model parameters based on recorded data. Model parameter estimation can be performed either by the least square method or by the maximum likelihood method. For these two methods, it is difficult to evaluate analytically the optimum values of the model parameters for a given data set and parameter estimation is rather performed by using classical numerical techniques.

In the following, the maximum likelihood method for estimating the unknown parameters θ, p_{sup} and p_{inf} is developed. Two types of failure data are considered: failure intervals (number of executions between failures) or numbers of failures per interval.

4.3.1. Estimation based on failure intervals

Let us assume that r software failures have been observed with m_1, m_2,..., m_r the times of failures occurrence. The likelihood function associated to (m_1, m_2,..., m_r) is the joint probability mass function f(m_1, m_2,..., m_r). Given that the probability of failure at execution P(i) is supposed to be independent of failure occurrences during the last i-1 executions, the likelihood function associated to
(m_1, m_2,..., m_r) is defined as:

$$L = f(m_1, m_2, \ldots\ldots, m_r) = \prod_{i=1}^{r} f(n_i \mid m_{i-1}) \quad (18)$$

with n_i the number of executions performed between the occurrence of failures i-1 and i. Using (6) and (8), we obtain:

$$L = \left\{P(m_1) \prod_{j=1}^{m_1-1} [1-P(j)]\right\} * \left\{P(m_2) \prod_{j=m_1+1}^{m_2-1} [1-P(j)]\right\} * \ldots * \left\{P(m_r) \prod_{j=m_{r-1}+1}^{m_r-1} [1-P(j)]\right\}$$

$$= \prod_{j=1}^{m_r} [1-P(j)] \prod_{j=1}^{r} \frac{P(m_j)}{1-P(m_j)} \quad (19)$$

An estimate of θ, p_{sup} and p_{inf} can be found by maximizing numerically the log-likelihood, i.e.,
$LL = \text{Ln } f(m_1, m_2, \ldots\ldots, m_r)$.

$$LL = \sum_{j=1}^{r} \text{Ln}\left\{\theta p_{sup}(1-p_{sup})^{m_j-1} + \bar{\theta}\, p_{inf}(1-p_{inf})^{m_j-1}\right\} - \sum_{j=1}^{r-1} \text{Ln}\left\{\theta(1-p_{sup})^{m_j} + \bar{\theta}(1-p_{inf})^{m_j}\right\} \quad (20)$$

4.3.2. Estimation based on numbers of failures per interval

Suppose that the collected data are provided in the form of $\{(s_1,y_1), (s_2,y_2) \ldots (s_k,y_k)\}$ with y_i the cumulative number of failures observed after s_i software executions. In order to evaluate the likelihood function associated to the collected data, the joint probability mass function relative to the sample $\{(s_1,y_1), (s_2,y_2) \ldots (s_k,y_k)\}$ needs to be calculated. Therefore it is necessary to calculate the probability mass function associated to the random variable Y_n defined previously. Using the approximation of the distribution of Y_n by a Poisson law, we obtain:

$$L = f\{(s_1,y_1), (s_2,y_2) \ldots (s_k,y_k)\} = \exp(-H(s_k)) \prod_{i=1}^{k} \frac{\{H(s_i) - H(s_{i-1})\}^{y_i - y_{i-1}}}{(y_i - y_{i-1})!} \quad (21)$$

An estimate of the model parameters can be obtained by maximizing the log-likelihood function given by:

$$LL = \sum_{i=1}^{k} (y_i - y_{i-1}) \text{Ln}\{H(s_i) - H(s_{i-1})\} - \text{Ln}\{(y_i - y_{i-1})!\} - H(s_k) \quad (22)$$

4.4. Application to field data

In order to illustrate the hyperexponential model application to field data, two failure data sets that are already reported in the literature will be considered. They will be referred to as data set A [31] and data set B [28] respectively. The model will be applied in order to follow up the evolution of cumulative number of failures.

4.4.1. Data set A

The failure data considered is sampled from the observed curve of cumulative number of failures presented in [31]. These data were obtained during testing of an application program written in PL/I and in Assembler language consisting of approximately 50,000 lines of

code. For the considered period, 773 executions were performed during which 73 failures were observed. The set of failure data obtained is presented in Fig. 5 where y_i is the cumulative number of failures observed after s_i software executions.

i	s_i	y_i
1	14	5
2	28	8
3	57	18
4	71	20
5	114	27
6	143	29
7	186	31
8	243	39
9	286	42
10	300	47
11	358	52
12	393	53
13	457	60
14	571	63
15	600	66
16	743	69
17	758	71
18	773	73

Figure 5: Failure data set A.

Figure 6 shows the application of the discrete time hyperexponential model to the considered data in order to evaluate and predict the cumulative number of failures; where:

- C0 is the observed cumulative number of failures,
- C1 the cumulative number of failures estimated by the hyperexponential model; the model is applied in a retrodictive way by considering the whole data set,
- C2 the cumulative number of failures estimated by the model when applied in a predictive way: data collected between i=1 to i=11 (i.e., up to s_i=358) are used to estimate model parameters, then the model is applied for 1-step ahead predictions from i=12 to 18 (i.e., from s_i = 358 to s_i= 773).

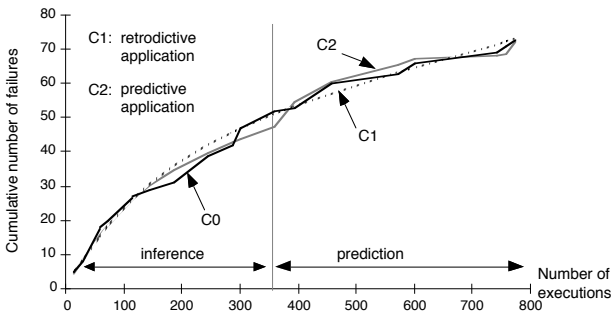


Figure 6: Model application to data set A

It can be seen that the model yields satisfactory estimations in both cases. To our knowledge, no statistical criteria for goodness-of-fit of discrete time reliability growth models to data are available in the literature; further investigations are needed to assess the validity of the results[3].

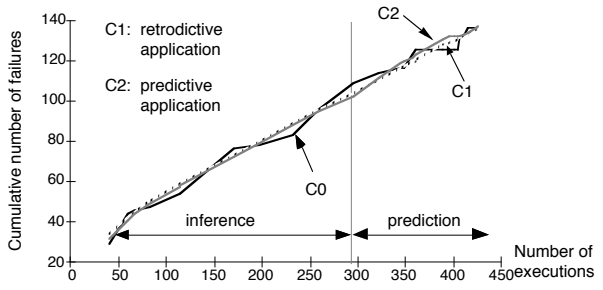
4.4.2. Data set B

The failure data reported in [28] and shown on Fig. 7 were recorded during combinational test of a real field software: 418 executions were performed during which 137 failures were observed.

i	s_i	y_i	i	s_i	y_i
1	33	30	14	341	117
2	51	45	15	349	124
3	59	47	16	354	127
4	74	48	17	381	127
5	105	55	18	387	127
6	106	55	19	393	127
7	163	77	20	397	127
8	190	79	21	398	132
9	225	84	22	403	134
10	251	96	23	408	137
11	287	110	24	416	137
12	315	115	25	418	137
13	337	117			

Figure 7: Failure data set B

Application of the hyperexponential model to this data in a similar way to the foregoing example leads to the results displayed on Fig. 8. Given that the slope of the observed cumulative number of failures curve is almost constant, it can be concluded that the software was in stable reliability and no significant reliability growth happened during the considered period. As shown by Fig. 8, stable reliability behavior can also be taken into account by the hyperexponential model as well as reliability growth behavior.



[3] Note that the $\chi 2$ statistical test usually used in these situations is better suited for comparing estimations derived from different models rather than for assessing the absolute validity of a given model.

Figure 8: Model application to data set B

5. Modeling the impact of the input probability distribution variation on software reliability

In previous sections, the software input data are supposed to be sampled according to an execution profile that is representative to the operational use of the system and software evaluation is performed without modeling explicitly the impact of execution profile features on software behavior. Nevertheless, many software systems are expected to be used with respect to different execution profiles. For example, consider a switching system that serves both urban and rural communities: the urban operational profile would contain a wider variety of types of calls such as credit card calls, conference calls, and international calls, in addition to residential calls [5]. Since the types of calls are different for the different field sites, it is expected that the software input domain will be covered with different probability distributions.

Given that the software execution profile may be subject to variations, then how to take into account different execution profiles when evaluating software reliability measures?

The execution profile of a software that is expected to be used with respect to k operational modes $\{OM_1,\ldots, OM_k\}$ can be defined by the input probability distribution $\{\pi_1, \ldots,\pi_k\}$, where π_i is the probability for the software to be used according to operational mode i.

We will assume that (Fig. 9):

- the reliability growth of the software when used with respect to each operational mode OM_i is characterized by a discrete time hyperexponential model $P_i(m_i)$ with parameters θ_i, $p_{i\cdot sup}$ and $p_{i\cdot inf}$: m_i is the number of executions performed with respect to OM_i,
- the software is observed during m_0 executions, with $m_0 = m_{01} + m_{02} + \ldots m_{0k}$ and m_{0i} is the number of software executions with respect to operational mode OM_i.

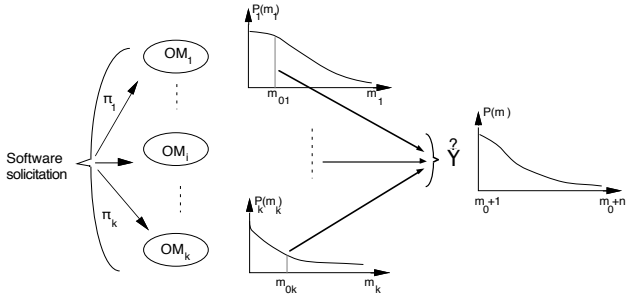


Figure 9: Reliability growth characterization considering different operational modes

Let n be the number of executions performed since m_0: $n = m-m_0$. We will investigate how to estimate and predict P(m), the software probability of failure at execution m, for $m > m_0$. Note that: $P(m) = P(n \mid m_0)$.

Two situations will be considered: i) the input probability distribution $\{\pi_1,\ldots,\pi_k\}$ remains invariant during software execution, and, ii) $\{\pi_1,\ldots,\pi_k\}$ varies after r executions.

5.1. Invariant input probability distribution

Consider first the case k=2. As after m_0, no information is available about the number of executions performed by the system with respect to each operational mode, $P(n \mid m_0)$ can be evaluated by taking the convolution of $P_1(n)$ and $P_2(n)$ within the interval $[m_0+1, m]$. We have:

$$P(m) = P(n \mid m_0) = \sum_{i=0}^{n-1} \binom{n-1}{i} \pi_1^{i} \pi_2^{n-1-i} \left\{\pi_1 P_1(m_{01}+1+i) + \pi_2 P_2(m_{02}+n-i)\right\} \quad (23)$$

It can be verified that:

$$\begin{cases} \pi_2=0 \text{ yields } P(n \mid m_0)=P_1(n \mid m_0) \\ \pi_1=0 \text{ yields } P(n \mid m_0)=P_2(n \mid m_0) \end{cases}$$

Relation (23), obtained for k = 2 can be generalized. It can be shown that for $k \geq 2$, we have:

$$P(m)=\prod_{l=1}^{k-1} \sum_{i_l=0}^{u_l} \binom{u_{l-1}}{i_{l-1}} \prod_{j=1}^{k} \pi_j^{i_j} \sum_{j=1}^{k} \pi_j P_j(m_{0j}+i_j+1) \quad (24)$$

with:

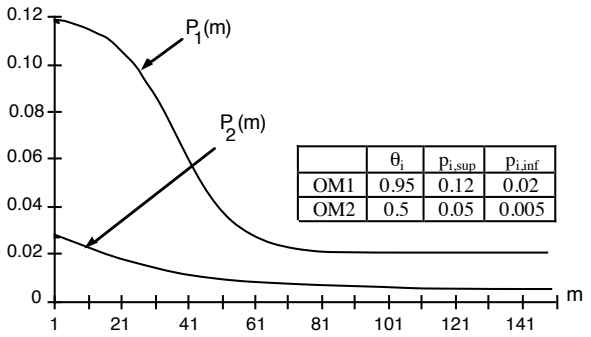
$$u_j = n - 1 - \sum_{s=1}^{j-1} i_s,\ j = 2, 3, \ldots k,\ u_1 = n-1 \text{ and } i_k = u_k \quad (25)$$

Note that the expression obtained for P(m) is relatively complex. Unfortunately, a simple form for P(m) does not

appear to be available, but a recursive scheme for the computation of (24) is easily arranged.

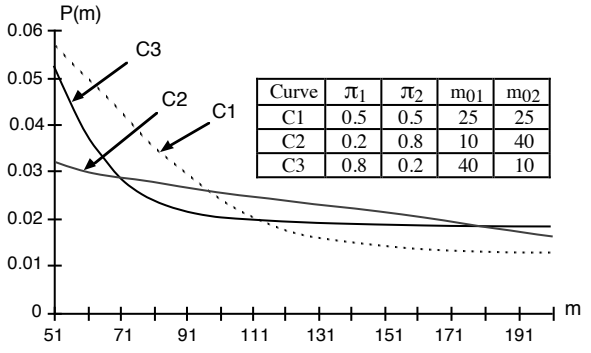
Figure 11 gives some examples of application of relation (23): two operational modes are considered and the probabilities of failure at execution of the software with respect to OM1 and OM2 are given by curves $P_1(m)$ and $P_2(m)$ displayed in Fig. 10. Given the assumed values for θ_i, $p_{i,sup}$ and $p_{i,inf}$ it can be seen that the software is assumed to be less reliable when used with respect to OM1 than to OM2.

Curves C1, C2 and C3 of Fig.11 plot the software probability of failure at execution, P(m), with respect to the number of executions m ($m>m_0=50$) for different values of the input probability distribution defined by $\{\pi_1, \pi_2\}$. Note that the same value m_0 is considered for the evaluation of C1, C2 and C3: $m_0=m_{01}+m_{02}=50$, nevertheless m_{01} and m_{02} depend on the probabilities π_1, and π_2. When m_0 is large enough, we have: $m_{0i} = \pi_i\, m_0$, i =1,2.



	θ_i	$p_{i,sup}$	$p_{i,inf}$
OM1	0.95	0.12	0.02
OM2	0.5	0.05	0.005

Figure 10: Probabilities of failure at execution with respect to OM1 and OM2



Curve	π_1	π_2	m_{01}	m_{02}
C1	0.5	0.5	25	25
C2	0.2	0.8	10	40
C3	0.8	0.2	40	10

Figure 11: Estimation of P(m) in a specified execution profile

As illustrated by Fig. 11, it can be seen that different reliabilities may be observed for the software depending on the values of π_1 and π_2 characterizing the execution profile according to which the software is used. Hence, a slight variation of the input probability distribution may lead to a significant change of the rate of decrease of P(m) as well as of its limiting value characterizing the stable reliability behavior of the software. Consider for instance curve C3, as the software is more frequently used according to operational mode OM1 than to OM2, then it can be seen that P(m) is similar to $P_1(m)$ given by Fig. 10 for $m > 50$. Hence, as shown by Fig. 11, P(m) decreases very quickly and reaches a stable reliability behavior characterized by:

$$P(\infty) = 0.8\, P_1(\infty) + 0.2\, P_2(\infty) \approx P_1(\infty)$$

However, for curve C2 a different behavior can be observed: P(m) decreases progressively and the stable reliability behavior is reached much more later in time.

Note that the software probability of failure at execution, P(m), corresponding to the limiting situations: $\pi_1=0$ and $\pi_2=0$ is given respectively by $P_2(m)$ and $P_1(m)$ displayed in Fig. 10.

5.2. Variant input probability distribution

Suppose now that input probability distribution $\{\pi_1, \pi_2, \ldots, \pi_k\}$ changes during the software execution. Such situations may occur for example when the software is released in an operational site in which the software is run with respect to an execution profile different from that used during software testing, or when different testing strategies are used during system testing for software validation.

Note by $\{\pi_1, \pi_2, \ldots, \pi_k\}_{\mathcal{A}}$ and $\{\pi_1, \pi_2, \ldots, \pi_k\}_{\mathcal{B}}$ two input probability distributions corresponding to two execution profiles $\mathcal{A}$ and $\mathcal{B}$. We will assume that execution profile change takes place after r software executions (Fig.12) and that execution profile variation does not modify the software probabilities of failure at execution with respect to operational modes OM_i; i.e., $P_i(m_i)$ is characterized by the same parameters θ_i, $p_{i,sup}$ and $p_{i,inf}$ before and after execution profile variation.

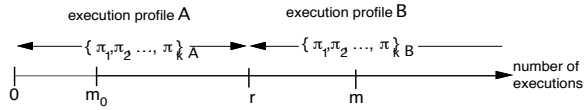


Figure 12: Execution profile variation

Using the same method as before, the probability of software failure at execution after execution profile variation can be derived.

Consider first the case k=2. It can be shown that, for $m \geq r$, we obtain:

$$P(m)=\sum_{i=0}^{n-1}\sum_{j=0}^{i}\binom{r-m_0}{j}\binom{m-r-1}{i-j}\pi_{1\mathcal{A}}^{j}\pi_{1\mathcal{B}}^{i-j}\pi_{2\mathcal{A}}^{r-m_0-j}\pi_{2\mathcal{B}}^{m-r-i+j-1} * \left\{\pi_{1\mathcal{B}}P_1(m_{01}+i+1)+\pi_{2\mathcal{B}}P_2(m_{02}+n-i)\right\} \quad (26)$$

For $m < r$, $P(m)$ is given by relation (23).

As **Erreur !**) = **Erreur !**= **Erreur !**, it can be verified that taking $\mathcal{A} = \mathcal{B}$ yields (23).

Generalization of relation (26) for $k \geq 2$ leads to:

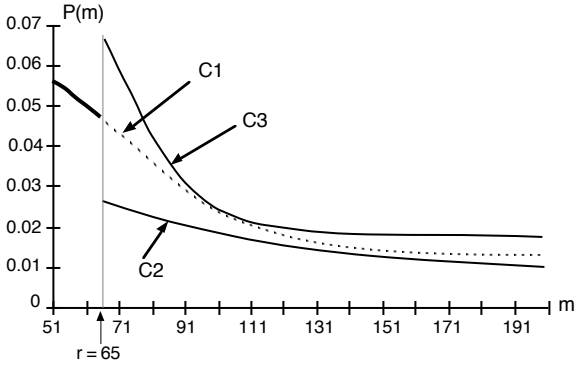
$$P(m)=\pi_{1\mathcal{A}}^{j_k}\pi_{1\mathcal{B}}^{i_k-j_k}\prod_{l=1}^{k-1}\sum_{i_l=0}^{u_l}\sum_{j_l=0}^{i_l}\binom{r-m_0-z_{l-1}}{j_l}\binom{m-r-1-v_{l-1}}{i_l-j_l} * \pi_{1\mathcal{A}}^{j_l}\pi_{1\mathcal{B}}^{i_l-j_l}\sum_{j=1}^{k}\pi_{j\mathcal{B}}P_j(m_{0j}+i_j+1) \quad (27)$$

with:

$$\begin{cases} s_l=\sum_{n=1}^{l-1} i_n \\ z_l=\sum_{n=1}^{l-1} j_n \\ u_l = n-1-s_l \\ v_l = s_l - z_l \text{ ;for } l = 2 \\ \dots \\ k; s_1 = 0 \\ z_1 = 0 \\ i_k = u_k \text{ and } j_k = r-m_0-z_k \end{cases}$$

In order to illustrate what precedes, a simple example is given in Fig. 13 for the case k=2:

- C1 shows the evolution of the software probability of failure at execution when the execution profile does not vary,
- C2 and C3 show the impact of execution profile variation on software probability of failure at execution.



Curve	$\pi_{1\mathcal{A}}$	$\pi_{2\mathcal{A}}$	$\pi_{1\mathcal{B}}$	$\pi_{2\mathcal{B}}$
C1	0.5	0.5	0.5	0.5
C2	0.5	0.5	0.2	0.8
C3	0.5	0.5	0.8	0.2

	θ_i	$p_{i,sup}$	$p_{i,inf}$
OM 1	0.95	0.12	0.02
OM 2	0.5	0.05	0.005

m_{01} m_{02} =25 r = 65

=

Figure 13: Impact on P(m) of execution profile variation

For $51 \leq m \leq 65$, C1, C2 and C3 cannot be dissociated as the software input probability distribution is assumed to be the same. The discrepancies observed between C1, C2 and C3 for $m>65$ are due to the change of the software execution profile. It can be seen that local variations of software reliability growth can be observed after execution profile change (at execution m=65): local decrease of P(m) for C2 and local increase for C3.

It is noteworthy that such discontinuities in the probability of failure at execution are likely to occur whenever the software execution profile is changed. When multiple execution profile variations occur, relation (27) has to be re-evaluated each time the software input probability distribution is changed in order to update the estimation of the software probability of failure at execution.

6. Conclusion

In this paper, a discrete time reliability growth model has been defined in order to estimate the software reliability growth behavior with respect to the number of executions performed. The proposed model is based on similar assumptions to those considered by continuous time reliability growth models based on non homogeneous Poisson processes. The software behavior is characterized by its probability of failure at execution which is supposed to be decreasing with the number of executions performed in order to take into account software reliability growth phenomenon resulting from the progressive removal of design faults. In order to cope with observed behavior of real life systems, it is assumed additionally that the

software reaches asymptotically a stable reliability behavior. The model is based on simple assumptions as no explicit relationships between fault removal process and software behavior are assumed. Explicit expressions for the main reliability measures are first derived and then some features of the model are illustrated through the application of the model to real life failure data. Furthermore, we have also discussed and illustrated the ability of the model to take into account explicitly the input probability distribution characterizing the execution profile in which the software is run in the evaluation of the probability of failure at execution. The modeling approach proposed is based on a simplifying assumption that is the software probability of failure at execution curve with respect to a given operational mode is not altered by the variation of the software execution profile. This assumption may not be always satisfied due, for instance, to the variation of load applied to the software [14]. Further work will consist in extending the results obtained in this paper to take into account these situations.

Due to space limitation, we restricted ourselves in this paper to discrete time reliability growth modeling of single component systems. Discrete time modeling of multi-component systems by the hyperexponential model is addressed in [8], [9].

Based on the features of the discrete time hyperexponential model, it can be seen that this model has similar properties to those of the continuous time hyperexponential model presented in [15]. The relationship between these models is discussed in [8] in which it is shown that the continuous time hyperexponential model can be deduced from the discrete time one if the software execution rate is taken into account in software reliability evaluation in addition to its probability of failure at execution.

Acknowledgments

The definition of the discrete time hyperexponential model presented in this paper was motivated by an innovative suggestion by Jean-Claude Laprie; the authors would like to express their thanks and appreciation. Thanks are also due to Alain Costes for his constructive comments on an earlier version of this paper.

This work has been partially supported by the CEC under ESPRIT Basic Research Action no. 3092 "Predictably Dependable Computing Systems (PDCS)"

References

[1] J.R. Brown, M. Lipow, "Testing for software reliability", *Proc.Int.Conf.Reliable Software,* pp.518-527.

[2] C.K. Cho, *Quality programming: developing and testing software with statistical quality control,* John Wiley & Sons, Inc., USA, 1987.

[3] T. Downs, "An approach to the modeling of software testing with some applications", *IEEE Trans. on Software Eng.,* vol. SE-11, no 4, Apr. 1985, pp. 375-386.

[4] J.W. Duran, J.J. Wiorkowski, "Quantifying software validity by sampling", *IEEE Trans. on Reliability,* vol. R-29, no. 2, June. 1980, pp. 141-144.

[5] W.K. Ehrlich, J.P. Stampfel, J.R. Wu, "Application of software reliability modeling to product quality and test process", *Proc. Int. Conf. on Software Eng.(ICSE),* Nice, May 1990, pp. 108-116.

[6] W. Feller, *An introduction to probability theory and its applications,* New York: John Wiley & Sons, 3rd Edition,1968.

[7] J.M. Finkelstein, "A logarithmic reliability growth model for single-mission systems", *IEEE Trans. on Reliability,* vol. R-32, no.5, Dec. 1983, pp. 508-511.

[8] M. Kaâniche, "Continuous time and discrete time hyper-exponential model for dependability growth modeling", Doctoral Thesis, Toulouse Polytechnic Institute, LAAS Rep. 92.002, Jan. 1992 (in French).

[9] M. Kaâniche, K. Kanoun, J.C. Laprie, "Discrete time reliability growth modeling of single and multicomponent software systems", LAAS Rep.92.046, Feb. 1992.

[10] J.D Kalbfleisch, R.L. Prentice, *The statistical analysis of failure time data,* John Wiley & Sons, 1980.

[11] K. Kanoun, T. Sabourin, "Software dependability of a telephone switching system", *Proc. 17th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-17),* Pittsburgh, June 1987, pp. 236-241.

[12] K.Kanoun, M.R Bastos Martini, J.M. de Souza, "A method for software reliability and prediction application to the TROPICO-R switching system", *IEEE Trans. on Software Eng.,* vol. SE-17, no. 4, Apr. 1991, pp. 334-344.

[13] J.C. Laprie, "Dependability modeling and evaluation of hardware-and-software systems",*Proc. 2nd GI/NTG/GMR Conf. on Fault Tolerant Computing,* Bonn, Germany, Sept. 1984, pp. 202-215.

[14] J.C. Laprie, "Hardware-and-software dependability evaluation",*Proc.IFIP 11th World Computer Congress*, San Francisco, USA, Aug 1989, pp.109-114.

[15] J.C. Laprie, K. Kanoun, C. Beounes, M. Kaâniche, "The KAT (Knowledge-Action-Transformation) approach to the modeling and evaluation of reliability and availability growth", *IEEE Trans. on Software Eng.*, vol. SE-17, Apr. 1991, pp. 370-382.

[16] B. Littlewood, "Forecasting software reliability", in *Software reliability modelling and identification*, Lecture notes in Computer Science, Springer-Verlag, Feb. 1989, pp 140-209.

[17] D.R. Miller, "Exponential order statistic models of software reliability growth", *IEEE Trans. on Software Eng.,* vol. SE-12, no. 1, Jan. 1986, pp. 12-24.

[18] J.D. Musa, K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement", *Proc. COMPSAC 84*, Chicago, 1984, pp. 230-238.

[19] J.D. Musa, A. Ianino, K. Okumoto, *Software reliability: Measurement, Prediction, Application*, McGraw-Hill, Singapore, 1987.

[20] E. Nelson, "A statistical basis for software reliability assessment", *TRW Software Series,* SS-73-03, March, 1973.

[21] E. Nelson, "Estimating software reliability from test data", *Microelectronics and Reliability,* 17, pp.67-74.

[22] D.L. Parnas, A.J. Van Schouwen, S. Po Kwan, "Evaluation of safety critical software", *Communications of the ACM*, Vol. 33, no. 6, June 1990, pp. 636-648.

[23] C.V. Ramamoorthy, F.B. Bastani, "Modeling of the software reliability growth process", *Proc. COMPSAC 80*, Chicago, Illinois, 1980, pp.161-169.

[24] C.V. Ramamoorthy, F.B. Bastani, "Software reliability—Status and perspectives", *IEEE Trans. on Software Eng.,* vol. SE-8, no. 4, July 1982, pp. 354-371.

[25] A.A. Salvia, R.C. Bollinger, "On discrete hazard functions", *IEEE Trans. on Reliability,* vol. R-31, no. 5, Dec. 1982, pp. 458-459.

[26] R.J. Serfling, "Some elementary results on Poisson approximation in a sequence of Bernouilli trials", *SIAM Review*, vol. 20, no 3, July 1978, pp. 567-579.

[27] Y. Tohma, K. Tokunaga, S. Nagase, Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hypergeometric distribution", *IEEE Trans. on Software Eng.*, vol. SE-15, Mar. 1989, pp. 345-355.

[28] Y. Tohma, H. Yamano, M. Ohba, R. Jacoby, "Parameter estimation of the hyper-geometric distribution for real test/debug data", *Proc. Int. Symp. on Software Reliability Engineering (ISSRE),* Austin, Texas, May 1991, pp. 28-34.

[29] S. Weiss, E.J. Weyuker "An extended domain based model for software reliability", *IEEE Trans. on Software Eng.,* vol. SE-14, no. 10, Oct. 1988, pp. 1512-1524.

[30] M. Xie, *Software reliability modeling*, World Scientific, Singapore, 1991.

[31] S. Yamada, S. Osaki, H. Narihisa, "Software reliability growth modeling with number of test runs", *Trans. IECE Japan,* vol. E-67, no. 2, Feb. 1984, pp. 79-83.