A Detailed Study of NHPP Software Reliability Models

(Invited Paper)

Richard Lai*, Mohit Garg
Department of Computer Science and Computer Engineering,
La Trobe University, Victoria, Australia

***Abstract*—Software reliability deals with the probability that software will not cause the failure of a system for a specified time under a specified condition. The probability is a function of the inputs to and use of the system as well as a function of the existing faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. Software Reliability Models (SRMs) provide a yardstick to predict future failure behavior from known or assumed characteristics of the software, such as past failure data. Different types of SRMs are used for different phases of the software development life-cycle. With the increasing demand to deliver quality software, software development organizations need to manage quality achievement and assessment. While testing a piece of software, it is often assumed that the correction of errors does not introduce any new errors and the reliability of the software increases as bugs are uncovered and then fixed. The models used during the testing phase are called Software Reliability Growth Models (SRGM). Unfortunately, in industrial practice, it is difficult to decide the time for software release. An important step towards remediation of this problem lies in the ability to manage the testing resources efficiently and affordably. This paper presents a detailed study of existing SRMs based on Non-Homogeneous Poisson Process (NHPP), which claim to improve software quality through effective detection of software faults.**

***Index Terms*—Software Reliability Growth Models, Non-Homogeneous Poisson Process, Flexible Models**

I. INTRODUCTION

Today, science and technology require high performance hardware and high quality software in order to make improvements and achieve breakthroughs. It is the integrating potential of the software that has allowed designers to contemplate more ambitious systems, encompassing a broader and more multidisciplinary scope, with the growth in utilization of software components being largely responsible for the high overall complexity of many system designs. However, in stark contrast with the rapid advancement of hardware technology, proper development of software technology has failed miserably to keep pace in all measures, including quality, productivity, cost and performance.

When the requirement for and dependencies on computers increase, the possibility of a crisis from computer failures also increases. The impact of failures ranges from inconvenience (e.g., malfunctions of home appliances), economic damage (e.g., interruption of banking systems), to loss of life (e.g., failures of flight systems or medical software). Hence, for optimizing software use, it becomes necessary to address issues such as the reliability of the software product. Using tools/techniques/methods, software developers can design/propose several testing programs or automate testing tools to meet the client's technical requirements, schedule and budget. These techniques can make it easier to test and correct software, detect more bugs, save more time and reduce expenses significantly [10]. The benefits of fault-free software to software developers/testers include increased software quality, reduced testing costs, improved release time to market and improved testing productivity.

There has been much effort expended in quantifying the reliability associated with a software system through the development of models which govern software failures based on various underlying assumptions [44]. These models are collectively called Software Reliability Models (SRMs). The main goal of these models is to fit a theoretical distribution to time-between-failure data, to estimate the time-to-failure based on software test data, to estimate software system reliability and to design a stopping rule to determine the appropriate time to stop testing and to release the software into the market place [4, 49]. However, the success of SRMs depends largely on selecting the model that best satisfies the stakeholder's need.

Recent research in the field of modeling software reliability addresses the key issue of making the software release decision, i.e., deciding whether or not a software product can be transferred from its development phase to operational use [8, 17, 50]. It is often a trade-off between an early release to capture the benefits of an earlier market introduction, and the deferral of product release to enhance functionality or improve quality. Despite various attempts by researchers, this question still stands and there is no stopping rule which can be applied to all types of data sets. Furthermore, hardly any work has been done on the unification of SRMs that can provide a solution for stakeholders to model and predict future failure behavior of a software system in a better way. Software reliability

*Corresponding author, E-mail: lai@cs.latrobe.edu.au

doi:10.4304/jsw.7.6.1296-1306

engineering produces a model of a software system based on its failure data to provide a measurement for software reliability. Several SRMs have been developed over the past three decades. As a general class of well developed stochastic process model in reliability engineering, Non Homogeneous Poisson Process (NHPP) models have been successfully used in studying hardware reliability problems. They are especially useful to describe failure processes which possess certain trends such as reliability growth and deterioration. Therefore, an application of NHPP models to software reliability analysis is easily implemented.

The mathematical and statistical functions used in software reliability modeling employ several computational steps. The equations for the models themselves have parameters that are estimated using techniques like least squares fit or maximum likelihood estimation. Then the models, usually equations in some exponential form, must be executed. Verifying that the selected model is valid for the particular data set may require iteration and study of the model functions. From these results, predictions about the number of remaining faults or the time of next failure can be made, and confidence intervals for the predictions can be computed.

A model is classified as an NHPP model if the main assumption is that the failure process is described by NHPP. Apart from their wide applicability in the testing domain, the main characteristic of this type of models is that there exists a mean value function which is defined as the expected number of failures up to a given time. In fact SRM is the mean value function of an NHPP. These models are flexible in nature as they can model both continuous and discrete stochastic processes. This paper presents a detailed study of existing SRMs based on Non-Homogeneous Poisson Process (NHPP), which claim to improve software quality through effective detection of software faults. The definitions, assumptions and descriptions of models based on NHPP will be provided, with the aim of showing how a large number of existing models can be classified into different categories.

II. The Spectrum of Software Reliability Models

The work on software reliability models started in the early 70's; the first model being presented in 1972. Various models proposed in the literature tend to give quite different predictions for the same set of failure data. It should be noted that this kind of behavior is not unique to software reliability modeling but is typical of models that are used to project values in time and not merely represent current values. Furthermore, a particular model may give reasonable predictions on one set of failure data and unreasonable predictions on another. Consequently, potential users may be confused and adrift with little guidance as to which models may be best for their applications. Models have been developed to measure, estimate and predict the reliability of computer software. Software reliability has received much attention because reliability has always had obvious effects on highly visible aspects of software development, testing prior to delivery and maintenance. Early efforts focused on testing, primarily because that is when the problems appeared.

As technology has matured, root causes of incorrect and unreliable software have been identified earlier in the life-cycle. This has been due in part to the availability of results from measurement research and/or the application of reliability models. The use of a model also requires careful definition of what a failure is. Reliability models can be run separately on each failure type and severity level. Reliability models are mathematically intense, incorporating stochastic processes, probability and statistics in their calculations, and relying on maximum likelihood estimates, numerical methods (which may or may not converge) and confidence intervals to model their assumptions.

Despite their shortcomings, such as excessive data requirements for even modest reliability claims, difficulty of taking relevant non-measurable factors into account etc. software reliability models offer a way to quantify uncertainty that helps in assessing the reliability of a software-based system, and may well provide further evidence in making reliability claims. According to the classification scheme proposed by Xie [44] considering the probabilistic assumption of SRM, and Kapur and Garg [17] considering the dynamic aspect of the models, the SRMs can be categorized into three categories viz. Markov, NHPP and Bayesian models. We briefly discuss the key features of Markov models and then study the NHPP and Bayesian models in detail.

A. Markov models

The Markov process represents the probabilistic failure process in Markov models. The software is represented by countable states, each state corresponding to a failure (fault). The main characteristic of such model is that the software, at a given point of time, has count ably many states and such states may be the number of remaining faults. Given that the process is at a specific state, its future development does not depend on its past history. The transition between the states depends on the present state of the software and the transition probability. The failure intensity of the software is assumed to be a discontinuous function which depends on the current state of the software.

Using this information, the Jelinski and Moranda (J-M) model [14] is modeled as a Markov process model. Next, Schick and Wolvertan [35] modified the J-M model by considering a time dependent failure intensity function and the time between failures to follow Weibull distribution. In addition, Shanthikumar [41] proposed a Markov model with time dependent transition probabilities. Then, Goel [6] modified the J-M model by introducing the concept of imperfect debugging. Later, Littlewood [25] proposed a model based on the semi-markov process to describe modular structured software.

- *Jelinski - Moranda De-eutrophication Model* - The J-M model is one of the earliest models for assessing software reliability by drawing inferences from failure data under some simple assumptions on the nature of the failure process. These assumptions are:

Assumptions	Reality
Faults are repaired immediately when discovered	Faults are not repaired immediately. A work-around may be to leave out duplicates and to accumulate test time if a non-repaired fault prevents other faults from being found. Fault repair may introduce new faults. It might be the case that newly introduced faults are less likely to be discovered as retesting is not as thorough as the original testing.
No new code is introduced in testing	It is frequently the case that fixed or new code is added during the test period. This may change the shape of the fault detection curve.
Faults are only reported by the testing group	Faults may be reported by lots of groups due to parallel testing. If the test time of other groups is added, there is a problem of equivalency between an hour of the testing group and an hour of other groups (types of testing may differ). Restricting faults to those discovered by the testing group eliminates important data.
Each unit of time is equivalent	The appropriate measure of time must relate to the test effort. Examples are: calendar time, execution time and number of test cases. However, potential problems are: the test effort is asynchronous (calendar time), some tests create more stress on a per hour basis (execution time) and tests do not have the same probability of finding a fault.
Tests represent operational profile	It is hard to define the operational profile of a product, reflecting how it will be used in practice. It would consist of a specification of classes of input and the probability of their occurrence. In test environments, tests are continually being added to cover faults discovered in the past.
Tests represent adoption characteristics	The rates of adoption, describing the number and type of customers who adopt the product and the time when they adopt, are often unknown.
Faults are independent	When sections of code have not been as thoroughly tested as other code, tests may find a disproportionate share of faults.
Software is tested in isolation	The software under testing might be embedded in a system. Interfaces with for example hardware, can hamper the measurement process (test delay due to mechanical or hardware problems, re-testing with adjusted mechanical or hardware parts).
Software is a black-box	There is no accounting for partitioning, redundancy and fault-tolerant architectures. These characteristics are often found in safety-critical systems.
The organization does not change	When multiple releases of a product are developed, the organization might significantly change, for example the development process and the development staff. After the first release, a different department might even execute the development of the next release. It may also heavily influence the test approach by concentrating on the changes made for corrective maintenance and preventive maintenance (a new functionality).

TABLE I. MODEL ASSUMPTIONS VS REALITY

1. At the beginning of testing, there are n_0 faults in the software code with n_0 being an unknown but fixed number.
2. Each fault is equally dangerous with respect to the probability of its instantaneously causing a failure. Furthermore, the hazard rate of each fault does not change over time, but remains constant at φ.
3. The failures are not correlated, i.e. given n_0 and φ the times between failures (Δt_1, $\Delta t2$, ..., Δt_{n0})
4. Whenever a failure has occurred, the fault that caused it is removed instantaneously and without introducing any new fault into the software.

$$z(\Delta t|t_{i-1}) = \phi[n_0 - M(t_{i-1}] = \phi[n_0 - (i-1)] \quad (1)$$

The failure intensity function is the product of the inherent number of faults and the probability density of the time until activation of a single fault, $n_a(t)$, i.e.:

$$\frac{dm(t)}{d(t)} = n_0[1 - \exp(-\phi t)] \quad (2)$$

Therefore, the mean value function is

$$m(t) = n_0[1 - \exp(-\phi t)] \quad (3)$$

It can easily be seen from equations (2) and (3) that the failure intensity can also be expressed as

$$\frac{dm(t)}{d(t)} = \phi[n_0 - m(t)] \quad (4)$$

According to equation (4), the failure intensity of the software at time *t* is proportional to the expected number of faults remaining in the software; again, the hazard rate of *n* individual faults is the constant of proportionality. Moreover, many software reliability growth models can be expressed in a form corresponding to equation (4). Their difference often lies in what is assumed about the per-fault hazard rate and how it is interpreted.

B. NHPP models

As a general class of well developed stochastic process model in reliability engineering, NHPP models have been successfully used in studying hardware reliability problems. These models are also termed as fault counting models and can be either finite failure or infinite failure models, depending on how they are specified. In these models, the number of failures experienced so far follows the NHPP distribution. The NHPP model class is a close relative of the homogenous poisson model, the difference is that here the expected number of failures is allowed to vary with time. Hence, they are useful for both calendar time data as well as for the execution time data.

C. Basic assumptions of NHPP models

Some of the basic assumptions (apart from some special ones for the specific models discussed) assumed for NHPP models are as follows:

1. A Software system is subject to failure during execution caused by faults remaining in the system.
2. The number of faults detected at any time is proportional to the remaining number of faults in the software.
3. Failure rate of the software is equally affected by faults remaining in the software.
4. On a failure, repair efforts starts and fault causing failure is removed with certainty.
5. All faults are mutually independent from a failure detection point of view.
6. The proportionality of failure occurrence/fault isolation/fault removal is constant.
7. Corresponding to the fault detection/removal phenomenon at the manufacturer/user end, there exists an equivalent fault detection/fault removal at the user/manufacturer end.
8. The fault detection/removal phenomenon is modeled by NHPP.

However, in practice, some of these assumptions may not hold their ground. Table 1 shows how assumptions and notions fail in reality [12, 26, 42, 43].

D. Comments on using NHPP models

Among the existing models, NHPP models have been widely applied by practitioners. The application of NHPP to reliability analysis can be found in elementary literature on reliability. The calculation of the expected number of failures/faults up to a certain point in time is very simple due to the existence of the mean value function. The estimates of the parameters are easily obtained by using either the method of maximum likelihood estimation (MLE) or least squares estimation (LSE).

Other important advantages of NHPP models which should be highlighted are that NHPPs are closed under super position and time transformation. We can easily incorporate two or more existing NHPP models by summing up the corresponding mean value functions. The failure intensity of the superposed process is also just the sum of the failure intensity of the underlying processes.

II. Relegation of NHPP Models

For binomial type there are a fixed number of faults at start, while for poisson type, the eventual number of faults could be discovered over an infinite amount of time. In poisson process models, there exists a relationship between:

- The failure intensity function and the reliability function
- The failure intensity function and the hazard rate
- Mean value function and cumulative distribution function (CDF) of the time to failure of an individual fault

If the mean value function *m(t)* is a linear function of time, then the process is the Homogeneous Poisson Process (HPP), however if it is a non-linear function of time, then the process is NHPP.

The earlier SRGMs, known as Exponential SRGMs, were developed to fit an exponential reliability growth curve. Similar to the J-M model [14], several other models that are either identical to the Exponential model except for notational differences or are very close approximations were developed by Musa [28], Schneidewind [36], and Goel and Okumoto [7]. Also, some Exponential models were developed to cater for different situations during testing [17, 45]. As a result, we have a large number of SRGMs each being based on a particular set of assumptions that suit a specific testing environment.

III. Model Groups

Generally, the SRGMs are classified into two groups. The first group contains models, which use machine execution time (i.e., CPU time) or calendar time as a unit of fault detection/removal period. Such models are called Continuous time models. The second group contains models which use the number of test cases as a unit of fault detection period. Such models are called discrete time models, since the unit of software fault detection period is countable. A large number of models have been developed in the first group while there are fewer in the second group. In this section, we explore a broad class of NHPP models based on Continuous and Discrete distributions. Table 2 categorizes commonly used NHPP models which show growth in reliability.

Model Group	Example
Continuous-time models	
- which use machine execution time (i.e. CPU time) or calendar time as a unit of fault detection/removal period	- Exponential model developed by Goel and Okumoto (G-O) [7] - Delayed S-shaped model due to Yamada et al. [46]
Discrete-time models	
- which use the number of test cases as a unit of fault detection period	- Exponential model developed by Yamada [47] - Delayed S-shaped model developed by Kapur et al. [17]

TABLE II. CONTINUOUS AND DISCRETE TIME MODELS

Notation	Description
a	Initial fault-content of the software
b	Fault removal rate per remaining fault per test case
a, b	Constants, representing initial fault content and rate of fault removal per remaining for a software
$m_f(t)$	Expected number of failures occurring in the time interval (0, t]

TABLE III. NOTATIONS OF EXPONENTIAL & DELAYED S-SHAPED CONTINUOUS-TIME MODELS

A. Continuous-time models

A very large number of Continuous time models have been developed in the literature to monitor the fault removal process which measure and predict the reliability of the software systems. During the testing phase, it has been observed that the relationship between the testing time and the corresponding number of faults removed is either exponential or S-shaped or a mix of two [1].

Let *[N(t), t≥ 0]* denote a discrete counting process representing the cumulative number of failures experienced (fault removed) up to time, *t*, i.e. *N(t)* is said to be an NHPP with intensity function λ(t), and it satisfies the following conditions:

1. There are no failures experienced at time *t = 0*, i.e. *N(t = 0) = 0* with probability 1.
2. The process has independent increment, i.e., the number of failures experienced in (t, t + Δt], i.e., N(t + Δt) − N(t), is independent of the history. Note this assumption implies the Markov property that the N(t + Δt) of the process depends only on the present state N(t) and is independent of its past state N(x), for x < t.
3. The probability that a failure will occur during (t, t+Δt] is λ(t)Δt+0(Δt), i.e., Pr[N(t+Δt)− N(t) = 1] = λ(t) + 0(Δt). Note that the function 0(Δt) is defined as:

$$\lim_{\Delta t\to\infty}\left(\frac{0(\Delta t}{\Delta t}\right)=0 \qquad (5)$$

In practice, it implies that the second or higher order effects of *Δt* are negligible.

4. The probability that more than one failure will occur during (t, t + Δt) is 0(Δt), i.e. Pr[N(t +Δt) − N(t) > 1] = 0(Δt).

Based on the above NHPP assumptions, it can be shown that the probability that *N(t)* is a given integer *k*, is expressed by:

$$\Pr[N(t)=k]=\frac{[m(t)]^k}{k!}\exp\{-m(t)\},k\ \geq 0 \qquad (6)$$

The function m(t) is a very useful descriptive measure of failure behavior. The function λ(t) which is called the instantaneous failure intensity is defined as:

$$\lambda(t)=\lim_{\Delta t\to 0}\left(\frac{\Pr[N(t+\Delta t)-N(t)>0]}{\Delta t}\right) \qquad (7)$$

given λ(t), the mean value function M(t) = ∑(N(t)] satisfies

$$m(t)=\int_0^t \lambda(S)ds \qquad (8)$$

Inversely, knowing m(t), the failure intensity function λ(t) can be obtained as:

$$\lambda(t)=\frac{dm(t)}{dt} \qquad (9)$$

Generally, by using a different non-decreasing function m(t), we obtain different NHPP models. Define the number of remaining software failures at time t by N(t) and we have that:

$$\bar{N}(t)=N(\infty)-N(t) \qquad (10)$$

where, N(∞) is the number of faults which can be detected by infinite time of testing. It follows from the standard theory of NHPP that distribution of $\bar{N}(t)$ is poisson with parameter [m(∞) − m(t)], that is:

$$\Pr[\bar{N}(t)=k]=\frac{[\mathrm{m}(\infty)-\mathrm{m}(\mathrm{t})]^k}{k!}\exp\{-m(\infty)-m(t)\}$$
$$k\ \geq 0 \qquad (11)$$

The reliability function at time t_0 is exponential, given by:

$$\mathrm{R}(\mathrm{t}|\mathrm{t}_0)=\exp\{-\mathrm{m}(\mathrm{t}+\mathrm{t}_0)-\mathrm{m}(\mathrm{t}_0)\} \qquad (12)$$

The above conditional reliability function is called a software reliability function based upon an NHPP for a Continuous SRGM.

- *Continuous-time exponential models* - G-O model [7] captures many software reliability issues without being overly complicated. It is similar to the J-M model except that failure rate decreases continuously in time. This is a parsimonious model whose parameters have a physical interpretation, and can be used to predict various quantitative measures for software performance assessment.

  According to basic assumption 8, m(∞) follows a poisson distribution with expected value N. Therefore, N is the expected number of initial software faults as compared to the fixed but unknown actual number of initial software faults n_0 in the J-M model.

Basic assumption 2 states that the failure intensity at time t is given by:

$$\frac{dm(t)}{dt}=\emptyset[N-m(t)] \qquad (13)$$

As in the J-M model, the failure intensity is the product of the constant hazard rate of an individual fault and the number of expected faults remaining in the software.
Following differential equation results from basic assumption 3:

$$m(t) = b(a - m(t)) \quad (14)$$

Solving the first order linear differential equation (14) with the initial condition m(t = 0) = 0 gives the following mean value function for NHPP:

$$m(t) = a(1 - exp(-bt)) \quad (15)$$

The mean value function given in equation (15) is exponential in nature and does not provide a good fit to the S-shaped growth curves that generally occur in software reliability. But the model is popular due to its simplicity.

- *Continuous-time delayed S-shaped model* - The model proposed by Yamada et al. [46] is a descendant of the G-O model [7], the data requirements being similar and the assumptions being similar with one exception. Yamada et al. reasoned that due to learning and skill improvements of the programmers during the debugging phase of the development cycle, the error detection curve is often not exponential but rather S-shaped.

$$\frac{dm_f(t)}{dt} = b\{a - m_f(t)\} \quad (16)$$

$$\frac{dm(t)}{dt} = b\{m_f(t) - m(t)\} \quad (17)$$

Solving equation (16) and (17) with initial conditions mf (t = 0) = 0 and m(t = 0) = 0, we obtain the mean value function as:

$$m(t) = a(1 - (1 + bt) \exp(-bt)) \quad (18)$$

alternatively the model can also be formulated a one-stage process directly as follows:

$$\frac{dm(t)}{dt} = b(t)(a - m(t) \quad (19)$$

Where $b(t) = \frac{b^2 t}{1+bt}$

It is observed that b(t) → b as t →∞. This model was specifically developed to account for lag in the failure observation and its subsequent removal. This kind of derivation is peculiar to software reliability only.

B. Discrete-time models

Yamada and Osaki [47] proposed two classes of Discrete Time Models. One class describes an error detection process in which the expected number of errors detected per test case is geometrically decreasing while the other class is proportional to the current error content. Kapur [17] proposed a discrete time model based on the concept that the testing phase has two different processes namely, fault isolation and fault removal. Kapur et al. [19] further proposed a discrete time model based on the assumption that the software consists of n different types of faults and each type of fault requires a different strategy to remove the cause of the failure due to that fault. Kapur et al. [18] also proposed a discrete time model with a discrete Rayleigh testing effort curve.

In addition to basic assumptions 1, 3 and 8, Kapur et al. [17-19] assumes the following for Discrete time models:

1. Each time a failure occurs, an immediate (delayed) effort takes place to decide the cause of the failure in order to remove it.

2. The debugging process is perfect - To obtain a realistic estimate of the residual number of faults, and the reliability, it is necessary to amend the assumption of instantaneous and perfect debugging. A number of researchers have recognized this shortcoming, and have attempted to incorporate explicit debugging into some of the software reliability models. Dalal [4] assumes that the software debugging follows a constant debugging rate, and incorporates debugging into an exponential order statistics software reliability model. Schneidewind [40], [39], [38] incorporates a constant debugging rate into the Schneidewind software reliability model [37]. Gokhale et al. [8] incorporates explicit repair into SRGM using a numerical solution. Imperfect debugging also affects the residual number of faults, and in fact at times can be a major cause of field failures and customer dissatisfaction. Imperfect debugging has also been considered by other researchers [7], [9-10].

During the software testing phase, software systems are executed with a sample of test cases to detect / remove software faults which cause software failures. A discrete counting process [N(n), n ≥ 0] is said to be an NHPP with mean value function m(n), if it satisfies the following conditions.

1. There are no failures experienced at n = 0, i.e. N(n = 0) = 0.

2. The counting process has independent increments, that is, for any collection of the numbers of test cases $n_1, n_2, ..., n_k$, where $(0 < n_1, < n_2 < ... < n_k)$.
The k random variable $N(n_1), N(n_2), -N(n_1), ..., N(n_k) - N(n_{k-1})$ are statistically independent.
For any number of test cases n_i and n_j , where $(0 \le n_i \le n_j)$, we have:

$$\Pr[N(n_j) - N(n_i) = x] = \frac{[m(n_j) - m(n_i)]^x}{x!} \exp\{-m(n_j) - m(n_i))\}, x \ge 0 \quad (20)$$

The mean value function m(n) which is bounded above and is non-decreasing in n represents the expected accumulative number of faults detected by n test cases. Then the NHPP model with m(n) is formulated by:

$$\Pr[N(n) = x] = \frac{[m(n)]^x}{x!} \exp\{-m(n)\}, x \ge 0 \quad (21)$$

Notation	Description
a	Initial fault-content of the software
b	Fault removal rate per remaining fault per test case
m(n)	The expected mean number of faults removal by the n^{th} test case
$m_f(n)$	The expected mean number of faults caused by the n^{th} test case

TABLE IV. NOTATIONS OF EXPONENTIAL & DELAYED S-SHAPED DISCRETE-TIME MODELS

As a useful software reliability growth index, the fault detection rate per fault (per test case) after the n^{th} test case is given by:

$$q(n) = \frac{[m(n+1)-m(n)]}{[m(\infty)-m(n)]}, n \geq 0 \quad (22)$$

where m( ) represents the expected number of faults to be eventually detected.
Let $\bar{N}$ (n) denote the number of faults remaining in the system after the n^{th} test case is given as:

$$\bar{N}(n) = N(\infty) - N(n) \quad (23)$$

The expected value of $\bar{N}$(n) is given by:

$$h(n) = m(\infty) - m(n) \quad (24)$$

which is equivalent to the variance of $\bar{N}$(n).
Suppose that n_d faults have been detected by n test cases. The conditional distribution of $\bar{N}$(n), given that N(n) = n_d, is given by:

$$\Pr\{\bar{N}(n) = y | N(n) = n_d\} = \frac{\{\Sigma(n)\}^y}{y!}\exp[-\{\Sigma(n)\}] \quad (25)$$

which means a poisson distribution with mean $\sum$(n), independent of n_d.

Now, the probability of no faults detected between the n^{th} and $(n + h)^{th}$ test cases, given that n_d faults have been detected by *r* test case, is given by:

$$R\left(\frac{h}{n}\right) = \exp[-\{m(n+h) - m(n)\}], n, h \geq 0 \quad (26)$$

The above conditional reliability function, called the software reliability function, is based on an NHPP for a Discrete SRGM and is independent of n_d.

- *Discrete-time exponential model* - Based on the previously mentioned assumptions for Discrete models, Yamada and Osaki [47] showed that the expected cumulative number of faults removed between the n^{th} and the $(n+1)^{th}$ test cases is proportional to the number of faults remaining after the execution of the n^{th} test run, and satisfies the following difference equation:

$$m(n+1) - m(n) = b(a - m(n)) \quad (27)$$

Solving the above difference equation using the probability generality function (PGF) with initial condition m(n = 0) = 0, one can obtain the closed form solution as:

$$m(n) = a(1 - (1 - b)^n) \quad (28)$$

The above mean value function is exponential in nature and does not provide a good fit to the S-shaped growth curves that generally occur in software reliability. Next, we briefly discuss below an S-shaped model.

- *Discrete-time delayed S-shaped model* - In the model developed by Kapur et al. [17], the testing phase is assumed to have two different processes namely, fault isolation and fault removal processes. Accordingly, we have two difference equations:

$$m_f(n+1) - m_f(n) = b(a - m_f(n)) \quad (29)$$

$$m(n+1) - m(n) = b(m_f(n) - m(n)) \quad (30)$$

Solving the above difference equation (29) and (30) using PGF with initial conditions m_f (n = 0) = 0 and m(n = 0) = 0 respectively, one can obtain the closed form solution as:

$$m(n) = a[1 - (1 + bn)(1 - b)^n] \quad (31)$$

Alternatively the model can also be formulated a one-stage process directly as follows:

$$m(n+1) - m(n) = \frac{b^2 n(n+1)}{1+bn}(a - m(n)) \quad (32)$$

It is observed that, $\frac{b^2 n(n+1)}{1+bn} \rightarrow$ b and n$\rightarrow \infty$. This model was specifically developed to account for lag in the failure observation and its subsequent removal.

IV. EXTENSIONS OF NHPP MODELS

Some NHPP models depict exponential reliability growth whereas others show S-Shaped growth, depending on the nature of growth phenomenon during testing. They are broadly classified into two categories. If the growth is uniform, generally Exponential models have been used and for non-uniform growth, S-shaped models have been developed. As S-shapedness in reliability can be ascribed to different reasons, many models exist in the literature, at times leading to confusion in model selection from the models available.

Initially, Goel and Okumoto [7] proposed the time dependent failure rate model based on NHPP. Later, Goel modified his original model [6] by introducing the test quality parameter. This is a continuous approximation to the original Exponential model and is described in terms of an NHPP process with a failure intensity function that is exponentially decaying. For all practical purposes, the

Model Type	Example
Flexible models	
- Models which can capture variability in either exponential and S-shaped curves	- Model for module structured software - Two types of fault models - To model the failure phenomenon of software during operation
Enhanced NHPP models	
- A model which incorporates explicitly the time-varying test-coverage function in its analytical formulation, and provides for defective fault detection and test coverage during the testing and operational phases.	- Log-logistic model
Time-dependent transition probability models	
- Models which can be used for both calendar time data as well as for the execution time data	- Basic execution model - Logarithmic model

TABLE V. EXTENSIONS OF NHPP MODELS

G-O and the other models are indistinguishable from the Exponential model. The Exponential model can be further generalized [13] to simplify the modeling process by having a single set of equations to represent a number of important models having the Exponential hazard rate function. The overall idea is that the failure occurrence rate is proportional to the number of faults remaining and the failure rate remains constant between failures while it is reduced by the same amount when a fault is removed.

In other cases, where there was a need to fit the reliability growth by an S-shaped curve, some available hardware reliability models depicting a similar curve were used [31]. In the literature, S-shapedness has been attributed to different reasons. Ohba and Yamada [33], and Yamada et al. [46] ascribed to it the mutual dependency between software faults whereas latter SRGMs were developed taking various causes of the S-shapedness into account, such as the models developed by Ohba [32], Yamada et al. [46], Kapur et al. [17], Kareer et al. [20], Bittanti et al. [3], Kapur and Garg [16], and Pham [34].

A. Flexible modeling approach

In addition to the models discussed above, other NHPP models termed as flexible growth models have been developed in the past which can capture variability in exponential and S-shaped curves. In this section, we present a brief overview of some of these models.

Ohba proposed a Hyper-Exponential model [31] to describe the fault detection process in module structured software; Khoshgoftaar [22] proposed the K-stage Erhangian model; Xie and Zhao [45] proposed a simple model with graphical interpretation; Kapur and Garg [15] modified the G-O model by introducing the concept of imperfect debugging; Zeephongsekul [49] proposed a model describing use when a primary fault introduces a secondary fault in real life software development projects. Non-uniform testing is more popular and hence the S-shaped growth curve has been observed in many software development projects.

Kareer et al. [20] and Yamada [48] proposed two types of fault models where each fault type is modeled by an S-shaped curve; Kimura et al. [23] proposed an exponential S-shaped model which describes software with two types of faults. Later in the testing phase, Kapur [17] ascribed it to the presence of different types of faults in software systems.

The above SRGMs have been proposed for the testing phase and it is generally assumed that the operational profile is similar to the testing phase, which may not be the case in practice. Very few attempts have been made to model the failure phenomenon of commercial software during its operational use. One of the reasons for this can be attributed to the inability of software engineering to measure the growth during the usage of software while it is in the market. This is unlike the testing phase where testing effort follows a definite pattern. Kenney [21] proposed a model to estimate the number of faults remaining in the software during its operational use. He has assumed a power function to represent the usage rate of the software, though he argues that the rate at which commercial software is used is dependent upon the number of its users. Kenney's model however fails to capture the growth in the number of users of the software.

Also, it is important that the SRGM should explicitly take into account faults of different severity. Such a modeling approach was earlier adopted by Kapur et al. [19]. This approach can capture variability in the growth curves depending on the environment in which it is being used and at the same time, it has the capability to reduce either exponential or S-shaped growth curves.

B. Enhanced NHPP models

The Enhanced NHPP model developed by Gokhale and Trivedi [11] states that the rate at which faults are detected is proportional to the product of the rate at which potential fault sites are covered and the expected number of remaining faults. This model allows for time-dependent failure occurrence rate per fault, i.e., the rate at which an individual fault will surface can vary with testing time.

The NHPP models have constant, increasing or decreasing failure occurrence rates per fault. These models were inadequate to capture the failure processes underlying some of the failure data sets, which exhibit an increasing/decreasing nature of the failure occurrence rate

per fault. The Log-logistic model was proposed to capture the increasing/decreasing nature of the failure occurrence rate per fault captured by the hazard of the Log-logistic distribution [24]. The mean value function m(t) in this case is given by:

$$m(t) = a\frac{(\lambda t)^k}{1+(\lambda t)^k} \quad (33)$$

where λ and *k* are constants.

C. Log-normal models

In his proposed model, Mullen [27] showed that the distribution of failure rates for faults in software systems tends to be lognormal. Since the distribution of event rates tends to be lognormal and faults are just a random subset or sample of the events, the distribution of the failure rates of the faults also tends to be lognormal.

The probability density function (pdf) of the lognormal distribution is given by:

$$dL(x) = \frac{1}{x\sigma\sqrt{2\pi}} exp^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}} dx (x > 0) \quad (34)$$

where, x is the variate, μ is the mean value of the log of the variate, and σ^2 is the variance of the log of the variate. The mean value of the variate is exp(μ+σ^2/2). The median value of the variate is exp(μ). The mode of the variate is exp(μ − σ^2) and its variance is exp(2μ + σ^2)exp(σ^2 − 1). If x is distributed as L(μ, σ^2) then 1/x is distributed as L(−μ, σ^2).

The cumulative distribution function (cdf) for the lognormal in the form of the tabulated integral of the standard normal density function is given as:

$$\int_0^x dL(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{(\ln(x)-\mu)}{\sigma}} exp^{-\frac{z^2}{2}} dz = \frac{\Phi((\ln(x)-\mu)}{\sigma} \quad (35)$$

The ability of the lognormal to fit the empirical failure rate distributions is shown to be superior to that of the gamma distribution (the basis of the Gamma/EOS family of reliability growth models) [2] or a Power-law model.

D. Time-dependent transition probability models

Some NHPP models are capable of coping with the case of non-homogeneous testing and hence they are useful for both calendar time data as well as for execution time data [44]. These models are termed Time-dependent transition probability models. In these models, the failure intensity decreases exponentially with the expected number of failures experienced. Musa [28] and Musa and Okumoto [30] proposed the basic execution time model based on the concept of failure observation and the corresponding fault removal phenomenon and log poisson model respectively.

- *Basic execution models* - This model is perhaps the most popular of the software reliability models [5]. The time between failures is expressed in terms of computational processing units (CPU) rather than the amount of calendar time that has elapsed. The model contains a feature for converting from calendar time to processing time or vice versa.

The mean value function is such that the expected number of failures is proportional to the expected number of undetected faults at that time i.e., the cumulative number of failures follows a poisson process.

$$m(t) = b_0(1 - exp^{-b_1 t}) \quad (36)$$

where, b_0, $b_1 > 0$

Musa himself [29] recommends the use of this model (as contrasted to Musa's logarithmic poisson model) when the following conditions are met:

a) Early reliability is predicted before program execution is initiated and failure data observed

b) The program is substantially changing over time as the failure data are observed

This model can also be used if one is interested in seeing the impact of a new software engineering technology on the development process.

- *Logarithmic poisson models* - This model [30] is similar to the G-O model except that it attempts to consider that later fixes have a smaller effect on a program's reliability than earlier ones. The model is also called Musa-Okumoto logarithmic Poisson model because the expected number of failures over time is a logarithmic function. Thus, the model is an infinite failure model.

The basic assumption of the model, beyond the assumption that the cumulative number of failures follows a poisson process, is that failure intensity decreases exponentially with the expected number of failures experienced:

$$m(t) = b_0 In(b_1 t + 1) \quad (37)$$

V. CONCLUSIONS

Reliability models are a powerful tool for predicting, controlling and assessing software reliability. As a general class of well developed stochastic process modeling in reliability engineering, NHPP models have been successfully used in studying hardware and software reliability problems. They are especially useful to describe failure processes which possess certain trends, such as reliability growth and deterioration, thus making the application of NHPP models to software reliability analysis easily implemented.

In this paper, we first studied the initial model (J-M) based on Markov process to provide a measurement for software reliability. These models were later grouped into NHPP and Bayesian models. We described the modeling process for both Continuous and Discrete time models based on NHPP. These models can also be classified according to their asymptotic representation as either concave or S-shaped. We explored a few commonly used extensions of NHPP models. Then, we studied the flexible modeling approach in which the models can be customized as per the need. Finally, we discussed Enhanced NHPP models and models based on time-

dependent transition probability to model both execution and calendar time. The existing NHHP SRGMs can help remove faults by accelerating the testing effort intensity, and the proper allocation and management of the testing resources.

A software release decision is often a trade-off between early release to capture the benefits of an earlier market introduction, and the deferral of product release to enhance functionality, or improve quality. In practice, software manufacturers are challenged to find answers to questions such as how much testing is needed?; how to manage the testing resources effectively and efficiently?; when should a product be released?; what is the market window?; what are the expectations of customers and end-users? etc. The decision making process to release a product will normally involve different stakeholders who will not necessarily have the same preferences for the decision outcome. A decision is only considered successful if there is congruence between the expected reliability level and the actual outcome, which sets requirements for decision implementation. NHHP SRGMs can help software practitioners decide if the reliability of a software product has reached a given threshold and to decide when the software system is ready for release.

REFERENCES

[1] Ch. A. Asad, M. I. Ullah, and M. J. Rehman. An Approach for Software Reliability Model Selection. International Computer Software and Applications Conference, (COMPSAC), pages 534–539, 2004.

[2] P. G. Bishop and R. E. Bloomfield. Using a log-normal failure rate distribution for worst case bound reliability prediction.

[3] S. Bittanti, P. Blozern, E. Pedrotti, M. Pozzi, and A. Scattolini. Forecasting Software Reliability. In G. Goss and J. Hartmanis, editors, A Flexible Modeling Approach in Software Reliability Growth, pages 101–140. Springer-Verlag, 1988.

[4] S. R. Dalal and C. L. Mallows. Some graphical aids for deciding when to stop testing software. IEEE Trans. on Software Engineering, 8(2):169–175, 1990.

[5] W. Farr. Software Reliability Modeling Survey. In M. R. Lyu, editor, Handbook of Software Reliability Engineering, pages 71–118. McGraw-Hill, Inc., 1996.

[6] A. L. Goel. Software Reliability Models: Assumptions, Limitations and Applicability. IEEE Transactions on Software Engineering, pages 1411–1423, 1985.

[7] A. L. Goel and K. Okumoto. Time-Dependent Error Detection Rate Model for Software Reliability and other Performance Measures. IEEE Transactions on Reliability, R-28(3):206–211, 1979.

[8] S. Gokhale, M. R. Lyu, and K. S. Trivedi. Analysis of software fault removal policies using a non homogeneous continuous time markov chain. Software Quality Journal, pages 211–230, 2004.

[9] S. Gokhale, P. N. Marinos, K. S. Trivedi, and M. R. Lyu. Effect of repair policies on software reliability. Proc. of Computer Assurance (COMPASS), pages 105–116, 1997.

[10] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi. Incorporating fault debugging activities into software reliability models: a simulation approach. IEEE Transactions on Reliability, 55(2):281–292, 2006.

[11] S. S. Gokhale and K. S. Trivedi. A Time/Structure Based Software Reliability Model. Annals of Software Engineering, 8:85–121, 1999.

[12] D. Hamlet. Are We Testing for True Reliability? IEEE Software, 9(4):21–27, 1992.

[13] American Institute of Aeronautics and Astronautics. Recommended Practice for Software Reliability. In ANSI/AIAA Report 0131992. AIAA, 1992.

[14] Z. Jelinski and P. B. Moranda. Software Reliability Research. In Statistical Computer Performance Evaluation (Ed.) W. Freiberger, pages 465–484, 1972.

[15] P. K. Kapur and R. B. Garg. Optimal Software Release Policies for Software Growth Model Under Imperfect Debugging. Researche Operationelle/Operations Research (RAIRO), 24:295–305, 1990.

[16] P. K. Kapur and R. B. Garg. A Software Reliability Growth Model for Error Removal Phenomenon. Software Engineering Journal, 7:291–294, 1992.

[17] P. K. Kapur, R. B. Garg, and S. Kumar. Contributions to Hardware and Software Reliability. World Scientific, Singapore, 1999.

[18] P. K. Kapur, M. Xie, R. B. Garg, and A. K. Jha. A Discrete Software Reliability Growth Model with Testing Effort. 1st International Conference on Software Testing, Reliability and Quality Assurance, 1994.

[19] P. K. Kapur, S. Younes, and S. Agarwala. A General Discrete Software Reliability Growth Model. International Journal of Modelling and Simulation, 18(1):60–65, 1998.

[20] N. Kareer, P. K. Kapur, and P.S. Grover. An S-shaped Software Reliability Growth Model With TwoTypes of Errors. Microelectronics Reliability, 30:1085–1090, 1990.

[21] G. Q. Kenney. Estimating Defects in Commercial Software During Operational Use. IEEE Transactions on Reliability, 42(1):107–115, 1993.

[22] T. M. Khoshgoftaar. Non-Homogeneous Poisson Process for Software Reliability. COMPSTAT, pages 13–14, 1988.

[23] M. Kimura, S. Yamada, and S. Osaki. Software Reliability Assessment for an Exponential S-shaped Reliability Growth Phenomenon. Computers and Mathematics with Applications, 24:71–78, 1992.

[24] L. M. Leemis. Reliability-Probabilistic Models and Statistical Methods. Prentice-Hall, 1995.

[25] B. Littlewood. Forecasting Software Reliability. In G. Goss and J. Hartmanis, editors, Software Reliability Modeling and Identification, chapter 5, pages 141–209. Springer-Verlag, 1987.

[26] H. Hecht M. Hecht, D. Tang and R. W. Brill. Quantitative Reliability and Availability Assessment for Critical Systems Including Software. 12th Annual Conference on Computer Assurance, pages 147–158, 1997.

[27] R. Mullen. The Lognormal Distribution of Software Failure Rates: Origin and Evidence. The Ninth International Symposium on Software Reliability Engineering (ISSRE), pages 124–133, 1998.

[28] J. D. Musa. A Theory of Software Reliability and its Applications. IEEE Transactions on Software Engineering, 1(3):312–327, 1975.

[29] J. D. Musa, A. Iannino, and K. Okumoto. Software Reliability: Measurement, Prediction, Application. McGraw-Hill, Inc., USA, 1987.

[30] J. D. Musa and K. Okumoto. A Logarithmic Poisson Execution Time Model for Software Reliability Measurement. International Conference on Software Engineering, (ICSE), pages 230–238, 1984.

[31] M. Ohba. Software Reliability Analysis Models. Nontropical Issue, pages vol. 28, Number 4, pp 428, 1984.

[32] M. Ohba. Inflection S-shaped Software Reliability Growth Model. In S. Osaki and Y. Hotoyama, editors, Lecture Notes in Economics and Mathematical System, pages 101–140. Springer-Verlag, 1988.

[33] M. Ohba and S. Yamada. S-shaped Software Reliability Growth Model. 4th International Conference on Reliability and Maintainability, pages 430–436, 1984.

[34] H. Pham. Handbook of Reliability Engineering. Springer-Verlag London limited, USA, 2003.

[35] G. J. Schick and R. W. Wolverton. An Analysis of Competing Software Reliability Models. IEEE Transactions on Software Engineering, 4(2):104–120, 1978.

[36] N. F. Schneidewind. Analysis of Error Processes in Computer Software. Sigplan Notices, 10:337–346, 1975.

[37] N. F. Schneidewind. Software reliability model with optimal selection of failure data. IEEE Trans. On Software Engineering, 19(11):1095–1014, 1993.

[38] N. F. Schneidewind. Modeling the fault correction process. Proc. of Intl. Symposium on Software Reliability Engineering (ISSRE), pages 185–191, 2001.

[39] N. F. Schneidewind. An integrated failure detection and fault correction model. Proc. of Intl. Conference on Software Maintenance, pages 238–241, 2002.

[40] N. F. Schneidewind. Assessing reliability risk using fault correction profiles. Proc. of Eighth Intl. Symposium on High Assurance Systems Engineering (HASE), pages 139–148, 2004.

[41] J. G. Shanthikumar. Software Reliability Models: A Review. Microelectronics Reliability, 23:903–949, 1983.

[42] J. A. Whittaker. What Is Software Testing? And Why Is It So Hard? IEEE Software, pages 70–79, 2000.

[43] A. Wood. Software Reliability Growth Models: Assumptions Vs. Reality. International Symposium on Software Reliability Engineering (ISSRE), pages 136–143, 1997.

[44] M. Xie. Software Reliability Modeling. World Scientific, Singapore, 1991.

[45] M. Xie and M. Zhao. On Some Reliability Growth Models With Simple Graphical Interpretations. Microelectronics Reliability, 33(2):149–167, 1993.

[46] S. Yamada, M. Ohba, and S. Osaki. S-shaped Reliability Growth Modeling for Software Error Detection. IEEE Transactions on Reliability, R-32:475–478, 1983.

[47] S. Yamada and S. Osaki. Discrete Software Reliability Growth Models. Applied Stochastic Models and Data Analysis, 1:65–77, 1985.

[48] S. Yamada, S. Osaki, and H. Narihisa. Software Reliability Growth Models With Two Types of Errors. Researche Operationelle/Operations Research (RAIRO), 19:87–104, 1985.

[49] P. Zeephongsekul, C. Xia, and S. Kumar. A Software Reliability Growth Model Primary Errors Generating Secondary Errors under Imperfect Debugging. IEEE Transactions on Reliability, R-43(3):408–413, 1994.

[50] D. R. Jeske and X. Zhang. Some Successful Approaches to Software Reliability Modeling in Industry. The Journal of Systems and Software, 74:85–99, 2005.