# A INVESTIGATIVE STUDY OF SOFTWARE RELIABILITY MODELS

**[1]Manas Kumar Yogi, [2]K.Chandrasekar**
[1]Assistant Professor, [2]Assistant Professor
[1, 2] Computer Science Engineering Dept.
[1,2]Pragati Engineering College, Surampalem, A.P. India

***Abstract— the software reliability models help in analysis of software products and they are built of some assumptions which induce a certain degree of unreliability also. In this paper we have investigated the most popular reliability models and presented the pitfalls of each model but not before discussing their operating principle as well as robustness. Our paper serves as a readymade guide to software quality assurance engineers to refer which model they want to consider while practicing the reliability activity during the maintenance phase so as to refine the models and apply them again into their needs.***

***Index Terms—Software Reliability, Fault, Quality, Black box, White box***

## I. INTRODUCTION

In modern era, while designing complex systems, a huge amount of functionality is delegated to software. This is done due to the fact that software is an intellectual product, which cannot be bounded by the limits of the physical world in the sense that the corresponding hardware system would be. As software always operates in the context of a larger system, the dependability requirements on the system gets reduced to the software components and become the essential software depend-ability requirements. Software reliability is one of the important factors of software quality and system dependability. It is defined as the probability of failure-free software operation in a specified environment for a specified duration of time. A software failure occurs when the functionality of the software departs from its specifications, and it is the result of a software fault, a design defect, being produced by an input to the code during its execution.

Software reliability analysis is done at various stages during the process of engineering software, for a system, as an attempt to evaluate if the software reliability requirements have been met. The analysis results not only provide feedback to the designers but also become a measure of software quality. There are two activities related to software reliability analysis: estimation and prediction. In either activity, statistical inference techniques and reliability models are applied to failure data obtained from testing or during operation to measure software reliability. However, estimation is usually retrospective and it is performed to determine achieved reliability from a point in the past to the present time. The prediction activity, on the other hand, parameterizes reliability models used for estimation and utilizes the available data to predict future reliability. In general, software reliability models can be classified as being black box models and white box models. The difference between the two is simply that the white box models consider the structure of the software in estimating reliability, while the black box models do not work in that way. IN this paper, we have presented a concise description of some fundamental black box and white box software reliability models. In section 2, black box models and related terminology are introduced. Five models from this category: the Jelinski-Moranda de-eutrophication model, the Goel-Okumoto non-homogeneous Poisson process (NHPP) model, the Musa basic execution time model, the enhanced NHPP (ENHPP) model and the Littlewood-Verrall Bayesian model, are described along with their basic assumptions, data requirements and model output.

In section 3, we investigate white box models and a high level classification of these models. Then I describe two models in this category that appeared in the literature in the recent past: Krishna-murthy and Mathur's path-based model and Gokhale's state-based model.

In Section 4, we have concluded the investigation with some observations. At the end we have also attempted to identify general shortcomings of software reliability models and suggest avenues for further, possibly fruitful, research.

## II. BLACK BOX SOFTWARE RELIABILTY MODELS

In the software development process, it is very typical to end up with a product that has many design defects, i.e. faults, or generally known as bugs. For a certain input to the software these faults are activated, resulting in a deviation of the software behavior from its specified behavior i.e. a failure. When failures are detected through the testing process and the corresponding faults are located, then assuming that these faults are perfectly fixed, i.e. the process of fixing a fault did not introduce a new fault, software reliability increases. If the failure data is recorded either in terms of number of failures observed per given time period or in terms of the time between failures, statistical models can be used to identify the trend in the recorded data, reflecting the growth in reliability. Such models are known as software reliability growth models (SRGMs) or growth models. They are considered for activities like prediction and estimation of software reliability.

All the SRGMs we have discussed in this section are of the black box type since they only consider failure data, or metrics that are gathered if testing data are not available. Black box models do not consider the internal structure of the software in reliability estimation and are termed like that because they consider software as a monolithic entity, a black box.

In the next portions of this section, five SRGMs are presented. These are namely the Jelinski-Moranda de-eutrophication model, the Goel-Okumoto non-homogeneous Poisson pro-cess (NHPP) model, the Musa basic execution time model, the enhanced NHPP (ENHPP) model and the Littlewood-Verrall bayesian model.

### 2.1 Key Terms

Some of the terms used in relation to SRGMs are listed below. Other terms are also used in different models and are explained as they are encountered.

1. M(t)=The total number of failures experienced by time t.
2. μ(t)= Mean value function for an SRGM. This represents the expectation of the number of failures expected by time t as estimated by the model. So we get, μ(t) = E[M(t)].

3. λ(t)= Failure intensity, representing the derivative of the mean value function. There-fore λ (t) = μ '(t).
4. Z( Δt/ $t_{i-1}$)= Hazard rate of the software, which indicates the probability density of experiencing the ith failure at $t_{i-1}$ + t given that that (i-1)st failure occurred at $t_{i-1}$.
5. z(t)=Per-fault hazard rate, which represents the probability that a fault, that had not been activated so far, will cause a failure instantaneously when activated. This term is usually assumed to be a constant (ϕ) by many of the models.
6. N=Initial number of faults present in the software prior to testing.

Data that are generally supplied to SRGMs are either times between failures {t1, t2, t3 …} or the times at which failure occurred {t1, t2, t 3 …}. All the models presented here make a common assumption that the failures have no dependence on each other.

### 2.2 The Jelinski-Moranda Model

This model which was developed in 1972, the Jelinski-Mornada deeutrophication model (J-M) is one of the first soft-ware reliability models.

The model assumes that:

1. N initial faults in the code prior to testing are a fixed but known value.
2. Failures are not correlated and the times between failures are independent and Exponentially distributed random variables.
3. Fault removal on failure occurrences is instantaneous and does not inject any new faults into the software which is undergoing test.
4. The hazard rate z(t) of each fault is time invariant and a constant (ϕ). Moreover, each fault is equally likely to cause a failure.

The assumptions lead to the hazard rate Z( t/ $t_{i-1}$) after removal of the (i – 1)st fault being proportional to the number of faults remaining in the software (N – M(ti-1)). Consequently we have

$$Z(t/t_{i-1}) = \phi (N - M(t_{i-1})) \quad (1)$$

The mean value function and the failure intensity functions for this model turn out to be

$$\mu(t) = N(1 - e^{-\phi t}) \quad (2)$$

$$\lambda(t) = N\phi e^{-\phi t} = \phi (N - \mu(t)) \quad (3)$$

Software reliability obtained from this model can then be expressed as

$$R(t_i) = e^{-\phi (N-(i-1)) t_i} \quad (4)$$

The model requires the elapsed time between failures or actual failure times for estimating its parameters.

### 2.3 The Goel-Okumoto Model

The Goel-Okumoto (G-O) non-homogeneous Poisson process (NHPP) model has slightly diff-erent assumptions from the J-M model. The significant difference between the two is the

assumption that the expected number of failures observed by time t follows a Poisson distribution with a bounded and non-decreasing mean value function (t). The expected value of the total number

Of failures observed in infinite time is a finite value N. The model also makes the following assumptions :

1. The number of software failures that occur in (t, t +Δ t] is proportional to the expected number of undetected faults, N - μ(t).
2. The number of failures detected in the inter-failure intervals (0, $t_1$), ($t_1$, $t_2$), … , ($t_n$-1, $t_n$) is not correlated.
3. The per-fault hazard rate is time invariant and a constant (ϕ).
4. The fault removal process when failures are detected is instantaneous and perfect.

The assumptions result in the following mean-value function (and corresponding failure intensity function) to be developed for the expected number of failures observed by time t.

$$\mu(t) = N(1 - e^{-\phi t}) \quad (5)$$

$$\lambda(t) = N\phi e\text{-}\phi t = \phi (N - \mu(t)) \quad (6)$$

We can observe that for a Poisson process, the failure intensity equals the hazard rate and therefore we have the failure intensity for a fault equals the constant per-fault hazard rate ϕ. It is clearly seen from equations (2), (3) and (5), (6) that G-O model is mathematically equivalent the J-M model. The basic difference lies in the assumptions and the interpretation of what N is.

In the J-M model, N is known and fixed, whereas in the G-O model, N itself is an expectation.

The model requires failure counts in the testing intervals and completion time for each test period for parameter estimation.

### 2.4 Musa's basic execution time model :

Musa's basic execution time model is the first attempt at incorporating a time variant test effort

Into reflecting software reliability growth as testing is performed. This model is also an NHPP model and assumes that the number of failures observed by a time is finite and follows a Poisson distribution.

The important assumptions that this model makes are:

1. The execution times between failures are exponentially distributed.
2. The per fault hazard rate is constant (ϕ)

The model requires that actual times of the failure be recorded as these times are used in the calculation of model parameters. The failure intensity obtained by application of the model to the failure data is

$$(t) = NB\phi e - B\phi t \quad (7)$$

Where N has the same interpretation as in the G-O model. B represents the fault reduction factor, which is a constant relating the rate of fault correction to the hazard rate. Equation (7) can also be put into the form

$$(t) = \phi (N - (t)) = Kf (N - (t)) \quad (8)$$

Where ϕ is formulated as the product of the linear execution frequency f, and the fault exposure ratio K, which can be interpreted as the average number of failures occurring per fault remaining in the code during one linear execution of the program. The parameter is also assumed to be constant over time. Equation (8) suggests that Musa's model is mathematically the same as the G-O and the J-M models. The main difference being the formulation of the per-fault hazard rate ϕ.

**2.5 The enhanced NHPP model**

The enhanced NHPP (ENHPP) model is a unifying framework for finite failure NHPP models i.e. other NHPP models with bounded mean-value functions are special cases of the ENHPP model. The model explicitly incorporates time-varying test coverage and imperfect fault detection in its analytical formulation.

Test coverage for this model is defined as the ratio of the number of potential fault sites sensitized by a test to the total number of potential fault sites. Potential fault regions refer to "the program entities representing either structural or functional program elements whose sensitization is deemed essential towards establishing the operational integrity of the software product".

The model makes the following assumptions:

1. Faults are uniformly distributed over all potential fault sites.
2. The probability of detecting a fault when a fault site is sensitized at time t is $c_d(t) = K$, (a constant), the fault detection coverage.
3. Faults are fixed perfectly.

The mean value function for this model is developed as

$$\mu(t) = c(t)N \quad (9)$$

Where c(t) is the time variant test coverage function and N is number of faults expected to have been exposed at full coverage. This is differentiated from N, which is the expected number of faults to be detected after infinite testing time, perfect test and fault detection coverage. The failure intensity for this model then formalizes as

$$\lambda(t)=z(t)(N - \mu(t)) \quad (10)$$

Where $z(t) = c'(t).(1 - c(t)) - 1$ is the time variant per-fault hazard rate. The model permits the scenario of defective coverage to be incorporated in the reliability estimation. Various coverage function distributions result in the variations of the NHPP models i.e. the G-O model, the Yamada S-shaped model, etc. Reliability as deducted from this model is expressed as

$$R(t/s) = e^{-NK(c(s+t)-c(s))} \quad (11)$$

Where s is the time of last failure and t is the time measured from last failure. Grottke observed correctly, that the main advantage of this model was to serve as a unifying frame-work for NHPP models. The dependence of the per-fault hazard rate only on time-variant test coverage disregards other considerable factors such as the fact that full test coverage may not be successful in detecting all the faults and that failures may still occur without any gain in test coverage.

**2.6 Little wood – Verrall bayesian model:**

The models investigated in the previous sections all assume that failure data is available. They also apply classical statistical techniques like maximum likelihood estimation (MLE) in which model parameters are fixed but unknown and are estimated from the available data. The problem with such a technique is that model parameters cannot be estimated when failure data is unavailable. Even when few data are available, MLE techniques are not trustworthy since they can result in unstable or incorrect estimations.

The bayesian SRGM considers reliability growth in the context of both the number of faults that have been detected and the failure-free operation. Further, in the absence of failure data, bayesian models consider that the model parameters have a prior distribution, which reflects judgement on the unknown data based on history e.g. a prior version and perhaps expert opinion about the software.

The Littlewood – Verrall model is one example of a bayesian SRGM that assumes that times between failures are independent exponential random variables with a parameter $\xi$ i, i = 1,2, …, n which itself has parameters $\psi$ (i) and $\alpha$( representing the programmer quality and task difficulty) having a prior gamma distribution. The failure intensity as obtained from the model using a linear form for the $\psi$ (i) function is

$$\lambda(t)= (\alpha - 1)(N^2+2B\phi(\alpha - 1))^{-\frac{1}{2}} \quad (12)$$

Where B represents the fault reduction factor, as in Musa's basic execution time model. This model requires tune between failure occurrences to obtain the posterior distribution from the prior distribution.

To end this section, almost all of the fundamental (non-bayesian) black-box models and their variations can be generalized to the model form of Equation (3) where failure intensity is proportional to the number of remaining faults in the software.

**III.WHITE BOX SOFTWARE RELIABILITY MODELS**

White box software reliability models consider the internal structure of the software in the relia-bility estimation as opposed to black box models which only model the interactions of software with the system within which it operates. The contention is that black box models are inadequate to be applied to software systems in the context of component-based software, increasing reuse of components and complex interactions between these components in a large software system. Furthermore, proponents of white box models advocate that reliability models that consider component reliabilities, in the computation of overall software reliability, would give more realistic estimates.

The motivation to develop the so-called "architecture" based models includes development of techniques to analyze performability of software built from reused and commercial off-the shelf (COTS) components, performing sensitivity analyses i.e. studying the variation of application reliability with variation in component and interface reliability, and for the identification of critical components and interfaces .

In these white box models, components and modules are identified, with the assumption that modules are, or can be, designed, implemented and tested independently. The architecture of the software is then identified, not in the sense of the traditional software engineering architecture but rather in the sense of interactions between components. The interactions are defined as control transfers, essentially implying that the architecture is a control-flow graph where the nodes of the graph represent modules and its transitions represent transfer of control between the modules. The failure behavior for these modules (and the associated interfaces) is then specified in terms of failure rates or reliabilities (which are assumed to be known or are computed sepa-rately from SRGMs). The failure behavior is then combined with the architecture to estimate overall software reliability as a function of component reliabilities. The way in which the failure behavior is combined with the architecture suggests that three generic classes of white box software reliability models exist: path based models, state based models and additive models.

**3.1 Krishnamurthy and Mathur's path based model :**

The primary assumption in this model is that component reliabilities are known. The model averages path reliability estimates over all the test cases run on particular software to estimate software reliability. Path reliability is computed from the sequence of components across a path that is followed when a particular test from the set of test cases is executed. The "architecture" of the software is essentially the sequence of components along different paths obtained from exe-cution traces collected from testing or simulation of the software.

If each component has reliability Ri then assuming that each component fails independently, the path reliability Rp of a trace M(P, t) for a program P containing a sequence of components m executed against a test case t belonging to a test suite T is

$$Rp = \prod_{m \in M(P,t)} Ri \quad (13)$$

The overall system reliability Rsys is the average of all path reliabilities over the test suite T is

$$Rsys = \frac{\sum Rp}{|T|} \quad (14)$$

Intra-component dependencies and the effect of a large number of loops introduce the possibility that multiple occurrences of a component along a trace path are not independent. This is modeled by considering that multiple occurrences of a component along the same path is equivalent to having k > 0 occurrences where k represents the degree of independence. The larger the value of k the lower is the estimate of path and consequently system reliability.

Other path based models follow similar approaches and system reliability is computed in general by considering possible execution paths of a program either by testing, experimentally or algorithmically.

### 3.2 Gokhale's state based model :

The particular characteristic of this model is that it either assumes that component reliabilities are available or it determines component reliabilities using the ENHPP SRGM. Incidentally, the developers of this model and the ENHPP model are the same. Another assumption is that the application for which reliability is to be predicted is a terminating application. Gokhale et al. observe that if the control transfers between modules are assumed to be a markov process then the control-flow graph, describing the architecture, of the software can be directly mapped into a discrete-time or continuous-time markov chain, with a one-one correspondence between the architecture and the markov chain.

The transitions of the markov chain represent the transition probabilities between modules (pij) and the expected time spent in a module i per visit ti is computed as a product of the expected execution time of each block and the number of blocks in the module.

Component reliabilities are computed from the ENHPP model as

$$R_i = e^{-\int_0^{V_i T_i} \lambda_i(t)\,dt}$$

Where $V_i$ is the expected number of visits to module I and i(t) is the time dependent failure intensity and ViTi is the total expected time spent in a module. The reliability of the overall system is then computed as

$$Rsys = \prod_{i}^{n} R_i \quad (16)$$

Another category of white box models includes models known as additive models. These models concentrate on computing overall system reliability using component failure data and do not try to develop an architecture for the system. The overall system failure intensity is calculated as the sum of component failure intensities while assuming that component reliability can be estimated using the NHPP class of software reliability growth models.

## IV. CONCLUSION

As stated by experts, "There is no universally acceptable model that can be trusted to give accurate results in all circumstances; users should not trust claims to the contrary. Worse, we cannot identify a priori for a particular data source the model of models, if any, that will give accurate results; we simply do not under-stand which factors influence model accuracy". Often its observed that a group of growth models having same assumptions in their predictions disagree for the same set of failure data and it is also the case that all the models make the same wrong prediction. In such a scenario, the predictions from the models are inconclusive and might only be best used for current reliability estimation but they cannot be applied for prediction.

With regard to white box models, most models make the assumption that component reliabilities are available and ignore the issue of how they can be determined. This is still an open research issue. With scarcity of failure data in components, it is not always possible to employ SRGMs to estimate component reliabilities such as in Gokhale's state based model. Moreover, the assumption of independence between failures in components can be violated du-ring unit testing, which implies that a reliability growth model can no longer be used to determine component reliabilities. Inter-component dependence is assumed to be nonexistent in architecture based models, which does not seem to be a very realistic assumption. The problem arises when an interface causes error propagation between two components and causes failures in both components. This invalidates the assumption of independence in component and interface failures and the models are no longer applicable.

The main strength of architecture based models, especially of state based models, is that the supporting framework for reliability prediction can also be used for performance analysis, also for analysis of sensitivity of the software modules and in the identification of critical components.

Most models depend on the existence of failure data, the only exception being that of bayesian growth models that assume a prior distribution for the SRGM parameters. However, these models inherit the drawback from their inapplicability when software reliability is considered to be a function of the reliabilities of its components and interfaces. This appears to be the case with the increasing use of COTS in building software. Prediction of reliability at the testing stage disallows major feedback to the design process due to reason that testing is too far down the software engineering cycle in a conventional software development model.

We have a opinion that a combined framework that utilizes software metrics early during the software engineering cycle, like failure data, when available, process metrics and process history to repeatedly estimate or predict reliability would be of great value in the view of early validation of reliability requirements, for making software design tradeoffs and for evaluating software architectures. We have observed , no framework exists till date that produces a justifiable prediction of software reliability when data is less and refines the prediction when data does become considerably available. These are areas which can be further researched by practioners.

**REFERENCES**


**[1]** P.G. Bishop, R. Bloomfield, "A conservative theory for long term reliability growth prediction", IEEE Transactions on Reliability, Vol. 45, No.4, pp. 550-560,Dec. 1996.

**[2]** P.G. Bishop, R. Bloomfield, "Worst case reliability prediction on a prior estimate of residual defects", Proceedings of the 13th IEEE International Symposium on Software Reliability Engineering (ISSRE-2002), pp. 295 – 303, Nov. 2002

**[3]** S. Gokhale, T. Philip, P. Marinos, K. Trivedi, "Unification of finite-failure non-homogenous Poisson process models through test coverage", Proceedings of the 7th IEEE International Symposium on Software Reliability Engineering (ISSRE-96), Nov. 1996.

**[4]** S. Gokhale, W.E. Hong, K. Trivedi, J.R. Horgan, "An analytical approach to architecture based software reliability prediction", Proceedings of the 3rd IEEE International Computer Performance and Dependability Symposium, (IPDS-98),pp.13-22,1998.

**[5]** M. Grottke, "Software reliability model study", PETS Project Technical report A.2, University of Erlangen-Nuremberg, 2001.

**[6]** M. R. Lyu (Ed.), Handbook of Software Reliability Engineering, IEEE Computer Society Press, 1996.

**[7]** S. Krishnamurthy, A.P. Mathur, "On the estimation of reliability of a software system using reliability of its components", Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE'97), pp.146 – 155,Nov. 1997.

**[8]** H. Pham, Software Reliability, Springer-Verlag, 2000.