A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair

Swapna S. Gokhale, Teebu Philip, Peter N. Marinos, Kishor S. Trivedi*
Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708-0291
{ssg, teebu, pnm, kst}@ee.duke.edu

Abstract

This paper reviews existing Non-Homogeneous Poisson Process (NHPP) models and their limitations, and proposes a more powerful non-homogeneous Markov model of the fault detection/removal problem. In addition, this non-homogeneous Markov model allows for the possibility of a finite time to repair a fault and for imperfections in the repair process. The proposed scheme provides the basis for decision making both during the testing and the operational phase of the software product. Software behavior in the operational phase and the development test phase are related and the release time formulae are derived. Illustrations of the proposed model are provided.

Index Terms: Software reliability modeling; Test coverage; Markov chains

1 Introduction

The production of modern computer software is one of the most complex and unpredictable activities in industry. Software is an integral part of critical applications such as commercial avionics, banking, nuclear power generation, and medical instrumentation. A methodology for certifying software integrity is absolutely essential. There is a need for far more accurate and cost-effective software reliability models than those presently in use.

A software reliability model must account quantitatively for test coverage and also imperfections in the software repair process. Since software reliability estimates impact significantly the release time of a software product, and thus its development and maintenance costs, the model accuracy becomes crucial.

The Enhanced Non Homogeneous Poisson Process(ENHPP)[12] software reliability model, an extension of the popular NHPP software reliability models[3], accounts explicitly for test coverage and test quality.

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

Traditionally, software reliability models do not explicitly consider the process of software repair. Many models assume that there is no specific time or cost requirement to repair. This assumption clearly needs to be amended in order to present a more realistic software reliability model. In this paper, we present a non-homogeneous Markov software reliability model which allows for finite time to repair. This new model is based on the ENHPP model, and allows for predictions to be made about the operational phase metrics, such as reliability and availability. Stopping rules based on various release criteria can also be developed from the non-homogeneous Markov software reliability model.

An example is provided to demonstrate the model.

2 Background and Motivation

In general, **test coverage** is a measure of how well a test covers all the potential fault-sites in a software product under test. It should be obvious that how one defines a *potential fault-site* and how well such fault-sites are sensitized influence greatly the significance of this important metric. Potential fault-sites are introduced here to mean program entities representing either structural or functional program elements whose sensitization is deemed essential towards establishing the operational integrity of the software product.

Definition 2.1 : *(Test Coverage): Given a software product and its companion test set, one defines test coverage, $c(t)$, to be the ratio of the number of potential fault-sites sensitized by time t divided by the total number of potential fault-sites under consideration.*

As mentioned earlier, there are several definitions of test coverage but the one offered here is most general and easily adaptable to situations which may benefit from a specialized application of the concept. The definition allowed for the possibility of defective coverage[12]; that is, the possibility of having a subset of non-sensitizable potential fault-sites. The definition of potential fault-site allows us to introduce into the area of software reliability a number of well-developed analytical techniques and notions which have been successfully applied to reliability studies in hardware systems[14].

Experimental data such as those presented in Table 1 can be easily collected, using tools such as ATAC[5], and utilized to obtain the test coverage function, $c(t)$, by associating coverage information given in column-2 with the cumulative execution time given in column-5.

2.1 Existing Software Reliability Models

All the leading software reliability models which are widely used for software quality assessment share the Markov property[15].

Table 1: Experimental Data

1	2	3	4	5	6
Test Number	Mean Coverage	Clock time	Execution time	Cumulative execution time	Failure occurrence (y/n)
1	c_1	T_1	t_1	t_1	n(no)
2	c_2	T_2	t_2	$t_1 + t_2$	y(yes)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	c_i	T_i	t_i	$\Sigma_{j=1}^{i} t_j$	n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	n
n	c_n	T_n	t_n	$\Sigma_{j=1}^{n} t_j$	y

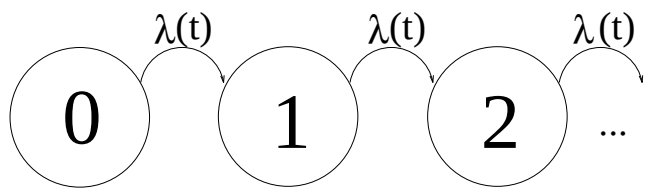

Figure 1: Non-homogeneous Markov chain for the NHPP and ENHPP models

Software faults are known to display the behavior of a Non-Homogeneous Poisson Process(NHPP) in which the parameter of the stochastic process, $\lambda(t)$, is time-dependent. The function $\lambda(t)$ denotes the instantaneous failure intensity. The Markov chain for the NHPP model is given in Figure 1, where $N(t)$ is the cumulative number of software faults detected by time t.

Given $\lambda(t)$, the mean value function $m(t) = E[N(t)]$ satisfies the relation,

$$m(t) = \int_0^t \lambda(s)ds \tag{1}$$

and,

$$\frac{dm(t)}{dt} = \lambda(t). \tag{2}$$

$N(t)$ as defined follows a Poisson distribution with parameter $m(t)$, that is, the probability that $N(t)$ is a given non-negative integer n is expressed as

$$P\{N(t) = n\} = \frac{[m(t)]^n * e^{-m(t)}}{n!}, n = 0, 1, 2, ...\infty \tag{3}$$

All the time domain models which assume the failure process to be a NHPP differ in the approach they use for determining $\lambda(t)$ or $m(t)$.

2.1.1 The Enhanced NHPP (ENHPP) Model

This reliability model[12] states that *the rate at which faults are removed is proportional to the rate at which potential fault-sites are covered.* This assumption is in contrast to the Goel-Okumoto(GO) model[3] which assumes that the fault removal rate is proportional only to the number of remaining faults in the software. The following basic assumptions are made for the Enhanced NHPP model:

- faults are uniformly distributed over all potential fault-sites,
- when a potential fault-site is sensitized, any fault present at that site is detected with probability $c_d(t)$, and
- repairs are effected instantly and without introduction of new faults.

Analytically, the model is based on the expression

$$\lambda(t) = \frac{dm(t)}{dt} = ac_d(t)\frac{dc(t)}{dt}. \tag{4}$$

or

$$m(t) = a\int_0^t c_d(\tau) * c'(\tau)d\tau \tag{5}$$

where a is defined as the total number of faults which are expected to be observed given infinite testing time, perfect fault detection coverage ($c_d(t) = 1$) and perfect test coverage($c(\infty) = 1$). The expected number of faults detected by time t is $m(t)$. If one assumes $c_d(\tau)$ to be a constant value K, then

$$m(t) = aKc(t). \tag{6}$$

Equation (6) is intuitively simple: the expected number of faults one should find by time t is equal to the total number of faults in the product times the probability of detecting a fault times the percent coverage gained by time t.

3 Equivalence of the Non-Homogeneous Markov Model and ENHPP

The ENHPP model provides a closed-form analytical expression for the expected number of faults. In this section, we demonstrate that the expected number of faults fixed at any given time t, i.e., $m(t)$, can also be numerically computed by solving the Markov chain shown in Figure 1 with SHARPE[13]. The chain in Figure 1 is infinite, but for practicality the chain is truncated to a states, where a is the expected number of faults found given perfect repair, perfect fault detection, perfect test coverage and infinite testing time. An estimate for a can be obtained by using the Maximum Likelihood Estimation (MLE) technique[12]. SHARPE is designed to solve homogeneous Markov chains, but we get around this problem by dividing the

time axis uniformly into small time intervals where within each interval, the failure intensity function of the ENHPP, $\lambda(t)$, can be approximated to be a constant. This value of $\lambda(t)$, is used to obtain the state probability vector of the Markov chain at the end of that time interval. These state probabilities will form the initial probability vector for the next time interval. Using the state probabilities, (probability of being in state i at time t is $p_i(t)$), we compute $m(t)$ with Equation (7).

$$m(t) = \sum_{i=0}^{a} i * p_i(t) \tag{7}$$

By using Markov chains, we can introduce sophistication into our model which could not be accounted for in the closed-form analytical approach such as:

- finite time to repair and
- imperfect repair

4 The Non-Homogeneous Markov Model with Finite Time to Repair

The NHPP based software reliability models assume instantaneous repair i.e., the fault that caused the software failure will be removed immediately. This assumption is clearly impractical. The time of correction of the fault, in general, does not coincide with the time of the original failure. This time-lag is not explicitly accounted for in the NHPP models, because it significantly complicates the failure process. The number of faults detected and fixed by a particular time will depend on the actual time taken to fix the defect and this number of faults will be less than the instantaneous repair case. In this model, we assume testing continues even during the repair process, and none of the faults are so severe that testing is rendered impossible. Therefore, faults will be queued until they are repaired.

The failure process is assumed to be unaffected by the repair mechanism, and the failure rate is given by the failure rate of the ENHPP model, as $\lambda(t)$. The repair rate will be assumed to be constant and equal to μ, leading to a mean repair time of $1/\mu$. The faults are repaired one at a time, and this implies that the detected faults form a queue up to a maximum of $a - l$, where l is the number of faults fixed. The state space for the Markov chain in this case is a tuple (i, j), where i is the number of faults fixed and j is the number of pending repairs. Thus the states denoted by the tuple $(i, 0)$, are the states with no pending repairs. Figure 2 shows the non-homogeneous Markov Chain with finite time to repair.

Computation of the expected number of faults fixed by time t, is given by Equation (8):

$$m(t) = \sum_{i=0}^{a} \sum_{j=0}^{a-i} i * p_{i,j}(t) \tag{8}$$

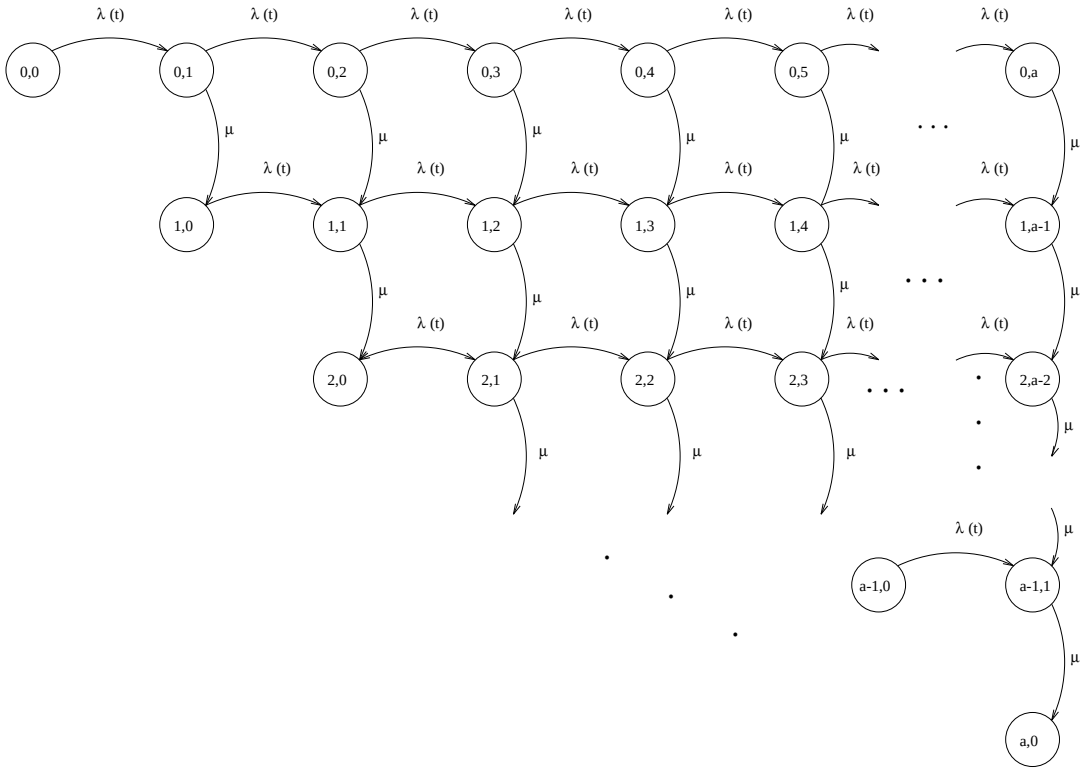

Figure 2: Non-homogeneous Markov chain with perfect repair

5 Non-Homogeneous Markov Model with Imperfect Repair

For NHPP and ENHPP models, the repair process has been assumed to be perfect and instantaneous. The instantaneous repair assumption was relaxed in the previous section. However, the process of fixing the defect was assumed to be perfect, and this may not always be the case. Here, we introduce the possibility of imperfect repair in the model, by noting that the fault which caused the failure may or may not be repairable and in the process of attempting a repair, new faults may be introduced in the software.

Let us initially assume that the repair mechanism is capable of removing a fault successfully with probability f, and in the process of doing so, it could introduce one or more faults in the software. We also assume that the number of faults introduced cannot exceed the total number of faults that have been detected and repaired at any given time. Thus at any state $(i,\ j)$, $(j \neq 0)$, the probability of introducing l faults $(l \leq i+j)$ is given by $b_{i+j,l}$. Thus we obtain Equation (9).

$$f + \sum_{l=1}^{i+j} b_{i+j,l} = 1 \tag{9}$$

The Markov chain for imperfect repair, assuming that at any state one may not introduce more than two faults, is shown in the Figure 3. Thus,

for state $(0,\ 1)$, $f_1 + b_{1,1} = 1$, and
for states $(i,\ j)$, $f + b_1 + b_2 = 1$, $i = 0$, $2 \leq j \leq a$.
for states $(i,\ j)$, $f + b_1 + b_2 = 1$, $1 \leq i \leq a$, $1 \leq j \leq a - i$.

The expression for the expected number of faults fixed remains the same as Equation (8).

The birth-death process used to model the software reliability by Kremer [9] and Kapur *et al* [8] is a special case of the non-homogeneous Markov model, if the number of faults introduced as a result of imperfect repair is restricted to one and the repair is instantaneous. If the failure rate per fault is assumed to be time-independent, the total failure rate is assumed to depend on the number of remaining faults and repair is instantaneous, our model yields the Goel-Okumoto imperfect debugging model[4], which was proposed as an extension to the Jelinski-Moranda [6] model.

6 Predictions in the Operational Phase

It has been widely stated that the operational profile of a software product is radically different from the profile used during product testing[7, 10]. A software reliability model must thus be able to make accurate predictions of the operational behavior of a software product.

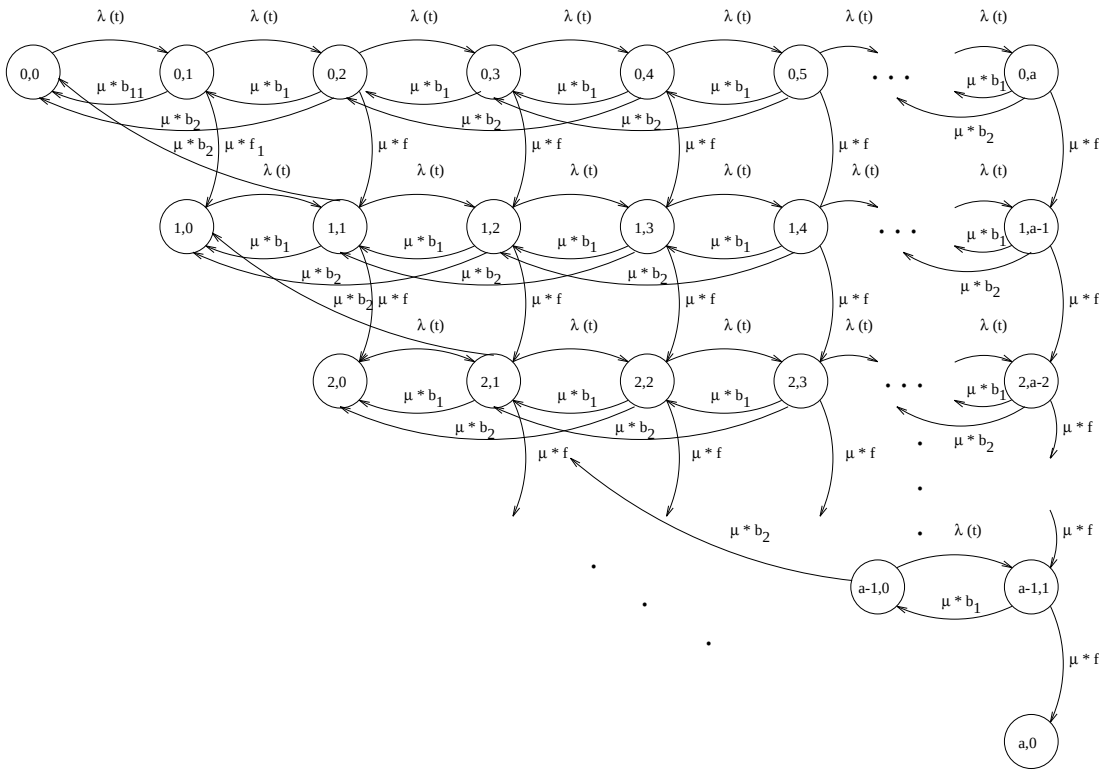

Figure 3: Non-homogeneous Markov chain with imperfect repair

To make accurate predictions in the operational phase, it becomes essential to provide estimates of the key parameters. The notions of test coverage, $c(t)$, and fault detection, $c_d(t)$, must be adjusted to reflect the user's operational environment and specialized needs.

Towards this end, it is first necessary to compute the number of remaining faults at the end of the development testing phase. If the software is released at time t_r, the expected number of faults fixed at time t_r is computed by SHARPE[13] using Equation (8). The expected number of remaining faults at the end of the development testing phase is thus given by

$$a_L = a - m_T(t_r) \tag{10}$$

Notice that in case of the ENHPP, Equation (10) was a closed-form analytical expression as a function of the release time t_r, where as in case of the non-homogeneous Markov model, this is a numerical value specific to a release time t_r.

The mean number of faults detected by time t in the operational phase is thus given by Equation(11)

$$m_L(t) = a_L * K_L * c_L(t) \tag{11}$$

where K_L is the fault detection probability and $c_L(t)$ is the coverage function during the operational phase. K_L and $c_L(t)$ capture the effects of the *operational profile* [7, 10, 11].

The failure intensity during the operational phase is given by Equation (12).

$$\lambda_L(t) = \frac{dm_L(t)}{dt} = a_L * K_L * c'_L(t) \tag{12}$$

The quantity $a - m_T(t_r) - m_L(\infty)$ denotes the number of faults which remain in the software product after infinite time of operational use. These faults may be the result of defective test coverage and imperfect fault detection in both the development testing and operational phases and imperfect repair in the development test phase. Conditioning on the faults that manifest during the system operation, we can derive the corresponding reliability function [15], for the operational phase to be:

$$R_L^c(t) = \frac{e^{-m_L(t)} - e^{-m_L(\infty)}}{1 - e^{-m_L(\infty)}} \tag{13}$$

The form of the equation is the same as in case of the ENHPP[12], and thus reliability prediction method in the operational phase is unaffected by the introduction of finite time to repair and imperfect repair in the development testing phase.

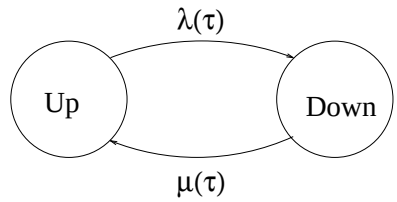

Figure 4: Two state model for the operational phase

7 Software Availability

In many applications, such as telecommunications software, software availability is a more critical system metric than reliability[2]. The expression for *software availability, A, during the operational phase*, is as follows:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{14}$$

where $MTTF$ is the mean time to failure and $MTTR$ is the mean time to repair. In this case, faults occurring during operation are not being fixed either because this is too expensive or because they are not easily reproducible to initiate an effective repair. Faults continue to reside in the software, and thus there is no opportunity for availability growth. $MTTR$ in this case is the mean time to reboot the system. The stochastic process capturing the behavior in the operational phase is thus a two state semi-Markov process as shown in Figure 4.

$MTTF$ is given by:

$$MTTF = \int_0^\infty R_L^c(t)dt = \int_0^\infty \frac{e^{-m_L(t)} - e^{-m_L(\infty)}}{1 - e^{-m_L(\infty)}} dt. \tag{15}$$

Assuming a special case of the coverage function during the operational phase, $c_L(t) = 1 - e^{-g_L t}$, the expression for $MTTF$ as in the case of ENHPP[12] is given as follows:

$$MTTF = \left(\frac{e^{-a_L K_L}}{g_L(1 - e^{-a_L K_L})}\right) * \sum_{i=1}^{\infty} \frac{(a_L K_L)^i}{i * i!} \tag{16}$$

8 Stopping Rules for Software Release Times

The critical issue that the software reliability models should address is *when to stop testing*. The determination of software release time is typically an optimization problem. The following stopping criteria can influence the software release times :

- *Stopping rule using the number of remaining faults.* This rule is typically applicable in the development phase. If testing is to stop when r out of a faults are detected and no repairs pending, with probability q, the stopping rule is given as :

$$\sum_{i=r}^{a} p_{i,0} \geq q \tag{17}$$

We are not guaranteed to achieve this criteria if the probability q, or the number of faults fixed r is required to be unreasonably high.

- *Stopping rule using failure intensity requirements.* The testing process is assumed to be independent of the repair mechanism, and hence the introduction of finite time to repair and imperfect repair strategies does not affect this stopping rule. Note that if the release time is t_r, then the failure intensity in the operational phase is $\lambda_L(t_r)$. Thus if the failure intensity in the operational phase is required to be $\lambda_L(t_r)$, the release time t_r can be determined as in the case of ENHPP[12] from Equation (18) which yields,

$$c_T(t_f) = \frac{\lambda_f * t_f}{aK_T} + C_0 \tag{18}$$

where $C_0 = c_T(0)$.

- *Stopping rule using reliability requirements.* If the required conditional reliability is R_r at time t_0 after product release, then the release time t_r is determined using Equation (13).

$$c_T(t_r) = \frac{\ln(1-(1-e^{-m_L(\infty)})(1-R_r)) + aK_L c_L(t_0)}{aK_T K_L c_L(t_0)} \tag{19}$$

- *Stopping rule using cost requirement.* Each state (i,j), can have a cost function associated with it, based on the number of faults fixed and number of pending repairs. Testing can thus stop at a time t_r when the total cost of testing exceeds a specified cost C. The stopping rule is thus given by the equation:

$$\int_0^{t_r} \sum_{i=0}^{a} \sum_{j=0}^{a-i} C_{ij} * p_{ij}(\tau) d\tau \geq C \tag{20}$$

This can be computed by discretizing the time interval $(0,t_r)$, and computing the state probability vector at the end of each interval as before. This state probability vector can then be used to calculate the cost of testing for that time interval, and thus the cummulative cost. This however, is a Markov reward model which can be solved directly by SHARPE[13], without having to calculate the state probability vectors.

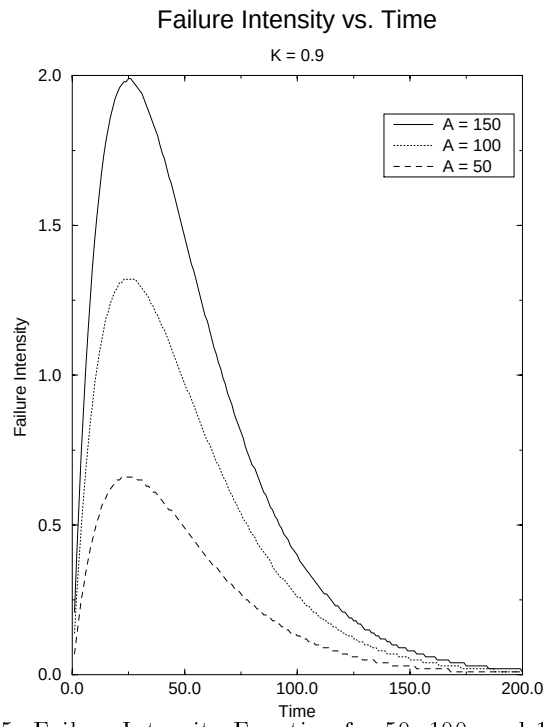

Figure 5: Failure Intensity Function for 50, 100, and 150 faults

- *Stopping rule based on availability* This rule is also defined as in case of the ENHPP[12]. Assuming that $MTTR$ is the time needed to reboot and there is no availability growth, the stopping rule based on availability can be derived by solving numerically Equation (21) for a_L and substituting into Equation (22)

$$(\frac{e^{-a_L K_L}}{g_L(1-e^{-a_L K_L})}) * \sum_{i=1}^{\infty} \frac{(a_L K_L)^i}{i * i!} = \frac{A_r * MTTR * g_L}{1 - A_r} \tag{21}$$

$$c_T(t_a) = \frac{a - a_L}{aK_T} \tag{22}$$

Note that the stopping rules based on the reliability, availability, and failure intensity criteria, will have some number of pending repairs unlike ENHPP, due to finite time to repair and imperfect repair. Also, the functions, $m_L(t)$ and a_L, are computed numerically now as opposed to closed-form expressions for the ENHPP model.

9 Illustration of the Model

We illustrate the model using two simple examples in this section. The MLE estimate for a is 50 in the first and 100 in the second. The detection probability is assumed to be constant and is equal to 0.9. The failure intensity function for $a = 50$, 100 and 150 faults is shown in the Figure 5.

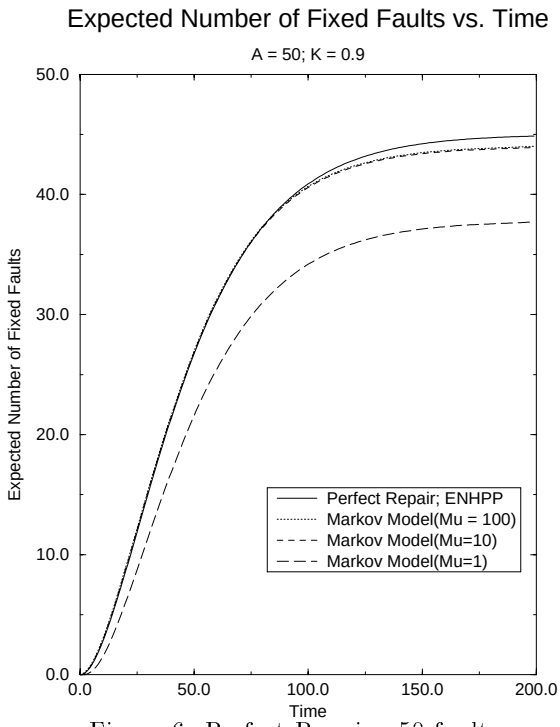

Figure 6: Perfect Repair - 50 faults

The failure intensity functions shown in Figure 5 were sampled at a time-interval of $t = 1$ unit, and these values were used in the non-homogeneous Markov Model. Thus for a given value of λ, the non-homogeneous Model becomes a homogeneous model and can be solved using SHARPE[13].

The expected number of fixed faults were computed with finite time to repair and imperfect repair. We can see that both in case of $a = 50$ and $a = 100$ faults, for any value of μ greater than $\lambda(t)$, the numerical Markov solution gives approximate answers close to the ENHPP model, as in the Figure 6 and Figure 7. Notice that the value of μ is very relevant in the expected number of fixed faults.

The expected number of fixed faults were computed for imperfect repair, with different values of μ. Notice that there is a significant difference in case of perfect and imperfect repair, as shown in Figures 8 and 9.

The expected number of faults were computed for imperfect repair, with different probabilities of the introduction of the fault for the same value of μ. As the probability of introduction of fault decreases, this approaches the case of perfect repair, as seen in Figures 10 and 11.

The release times were compared for $a = 50$ faults, for different values of k, for a threshold of $q = 0.8$. It can be seen from the Figure 12, that as k increases the release time increases. For the same value of k and the threshold, the release times are higher for imperfect repair than perfect repair, as shown in Figure 13 and Figure 14.

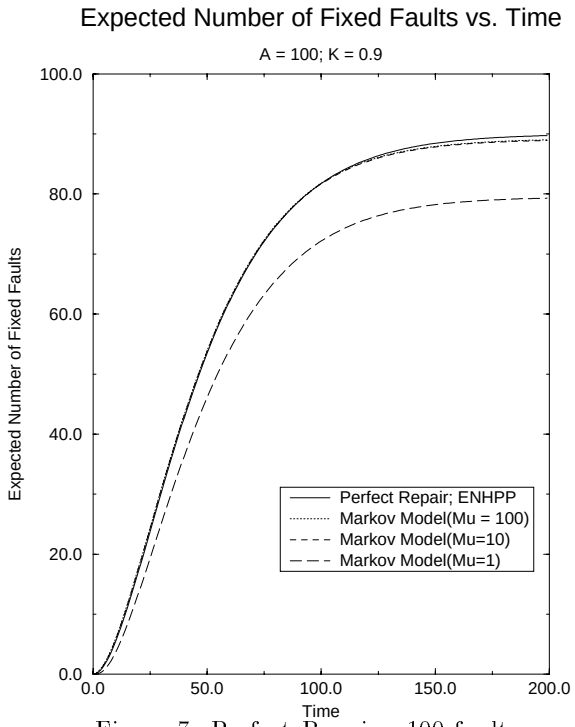

Figure 7: Perfect Repair - 100 faults

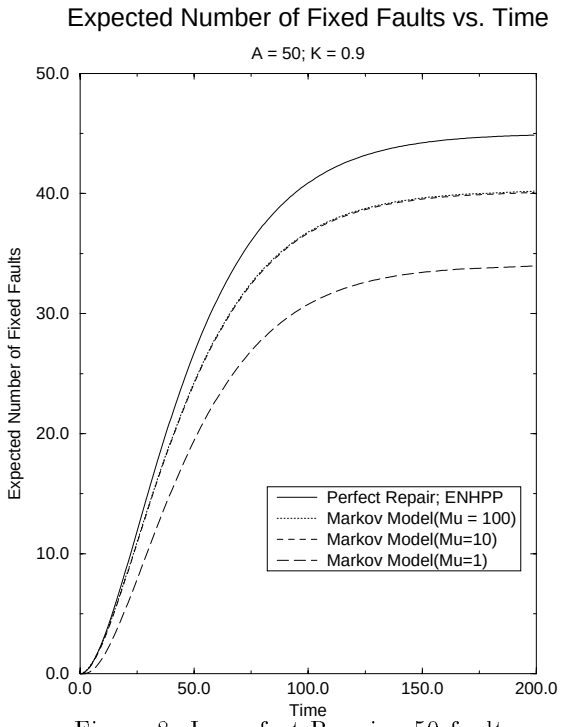

Figure 8: Imperfect Repair - 50 faults

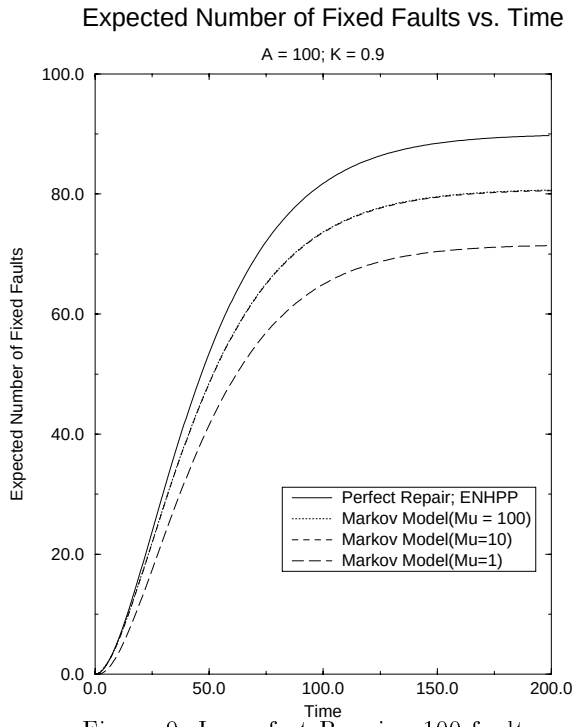

Figure 9: Imperfect Repair - 100 faults

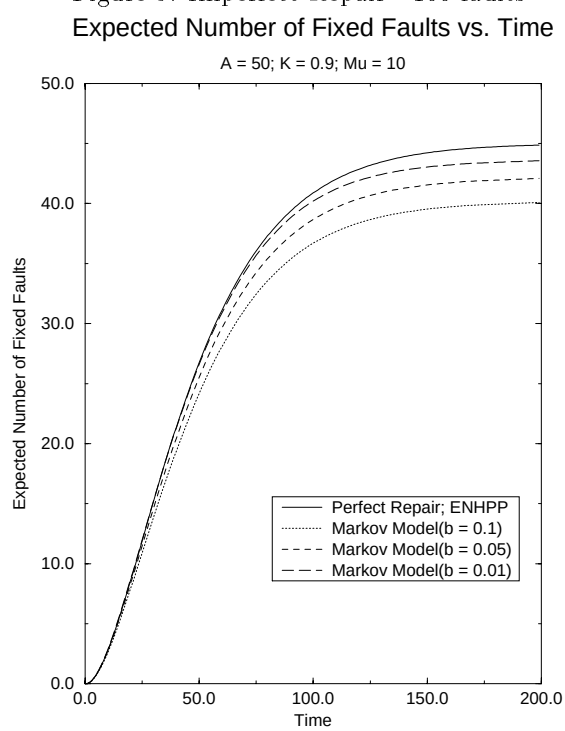

Figure 10: Imperfect Repair - 50 faults - Different fault probabilities

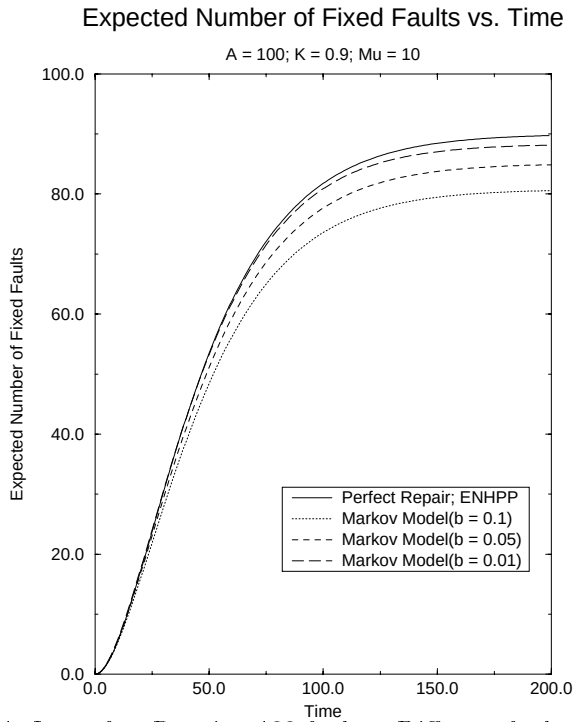

Figure 11: Imperfect Repair - 100 faults - Different fault probabilities

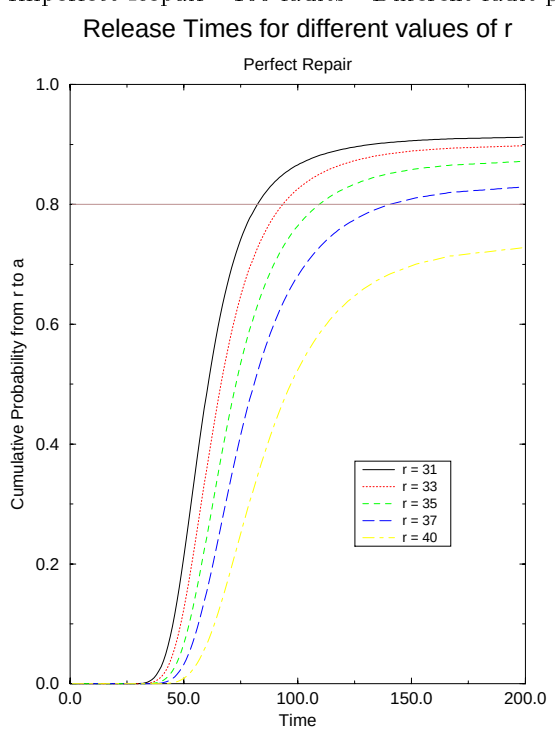

Figure 12: Release times based on a number of remaining faults for a = 50

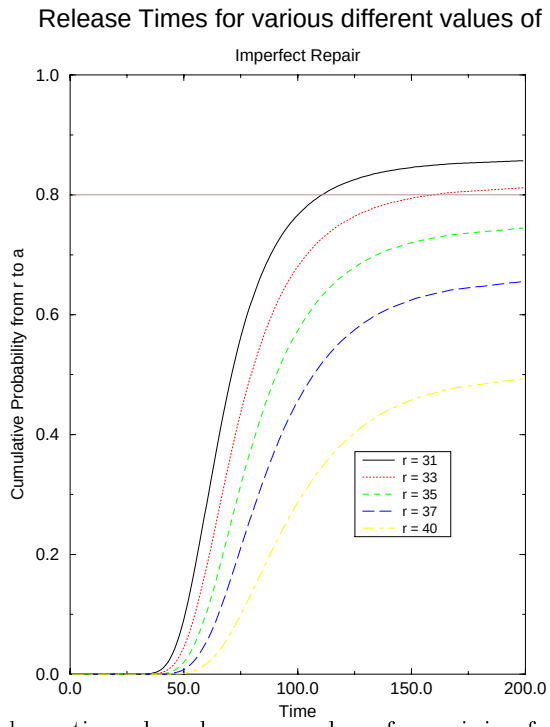

Figure 13: Release times based on a number of remaining faults for a = 50

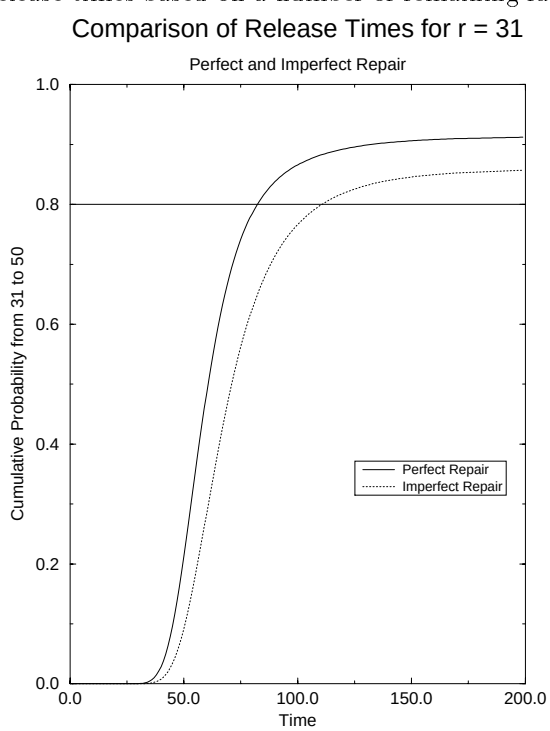

Figure 14: Release times for for 31 out of 50 faults

10 Conclusions

In this paper, we present the non-homogeneous Markov software reliability model based on the ENHPP model. The non-nomogeneous Model can handle the cases of finite repair time and imperfect repair. Numerical computation is required to solve for expected number of faults, reliability and availability. We use the SHARPE software tool for this purpose. Stopping rules based on the non-homogeneous Markov model are also developed.

Further areas of research include:

- Simplifying the state space of the non-homogeneous Markov model, to enable faster computations.
- Extending the non-homogeneous Markov Model to handle multiple defect types.
- Introducing the notion of multiphased execution in the non-homogeneous Markov Model, and
- Determining methods to validate the non-homogeneous Markov Model.

11 Acknowledgements

The authors would like to thank Linda Alger, Jay Lala, Paul Cefola and Ron Proulx of the Charles Stark Draper Laboratory for their valued input. The authors would also like to thank Warren Debany of the US Air Force Rome Laboratory for his technical assistance. We would also like to thank Lawrence J. Isley of IBM for his insightful discussions.

References

[1] P. Cefola, R. Proulx, R. Metzinger, M. Cohen and D. Carter, "The RADARSAT Flight Dynamics System: An Extensible, Portable, Workstation-based Mission Support System," *Proc. AIAA/AAS Astrodynamics Conference*, August 1994.

[2] R. DeMillo, Keynote address at *Fourth Bellcore/KPN/Purdue Workshop on Issues in Software Reliability,* Leidschendam, The Netherlands, October 1995.

[3] A.L. Goel and K. Okumoto, "Time-Dependant Error-Detection Rate Models for Software Reliability and Other Performance Measures," *IEEE Trans. on Reliability*, Vol. R-28, No. 3, pp. 206-211, August 1979.

[4] A.L Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. on Software Engineering*, vol. SE-11, No. 12, pp. 1411-1423, December 1985.

[5] J.R. Horgan, S. London, and M.R. Lyu, "Achieving Software Quality with Testing Coverage Measure," *IEEE Computer*, vol. 27, no. 9, pp. 60-69, September 1994.

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

[7] K. Kanoun and J.C. Laprie, "Software Reliability Trend Analysis: From Theoretical to Practical Considerations," *IEEE Trans. on Software Engineering*, vol. 20, no. 9, pp. 740-747, September 1994.

[8] P.K. Kapur, K.D. Sharma and R.B. Garg, "Transient Solutions of Software Reliability Model with Imperfect Debugging and Error Generation," *Microelectronics and Reliability*, vol. 32, no. 1, pp 475-478, April 1992.

[9] W. Kremer, "Birth-death and Bug Counting," *IEEE Trans. on Reliability*, vol. R-32, no. 1, pp. 37-47, April 1983.

[10] I. Lee and R.K. Iyer, "Software Dependability in the Tandem GUARDIAN System," *IEEE Trans. on Software Engineering*, vol. 21, no. 5, pp. 455-468, May 1995.

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

[12] T. Philip, P.N. Marinos, K.S. Trivedi and J.N. Lala, "A Multiphase Software Reliability Model: From Testing to Operational Phase", TR-96-01, Center for Advanced Computing and Communication, Duke University, January 1996.

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

[14] S.C. Seth, V.D. Agrawal and H. Farhat, "A Statistical Theory of Digital Circuit Testability," *IEEE Trans. on Computers*, Vol. 39, No. 4, pp. 582-586, April 1990.

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