Effect of Repair Policies on Software Reliability

Swapna S. Gokhale[1,*], Peter N. Marinos[1], Michael R. Lyu[2], Kishor S. Trivedi[1,†]

[1] Center for Advanced Computing & Communication
Dept. of Electrical and Computer Engg.
Duke University, Durham, NC 27708-0291
E-mail: {ssg,pnm,kst}@ee.duke.edu

[2] Room No. 2A-413, Lucent Technologies, Bell Laboratories
600, Mountain Avenue, Murray Hill, NJ 07974
E-mail: lyu@research.bell-labs.com

Abstract

Software reliability is an important metric that quantifies the quality of the software product and is inversely related to the number of unrepaired faults in the system. Fault removal is a critical process in achieving desired level of quality before software deployment in the field. Conventional software reliability models assume that the time to remove a fault is negligible and that the repair process is perfect. In this paper we examine various kinds of repair scenarios, and analyze the effect of these fault removal policies on the residual number of faults at the end of the testing process, using a non-homogeneous continuous time Markov chain. The fault removal rate is initially assumed to be constant, and it is subsequently extended to cover time and state dependencies. These fault removal scenarios can be easily incorporated using the state space view of the non-homogeneous Poisson process.

1 Introduction

The increase in the production and maintenance costs of software relative to those of hardware in computer systems have prompted considerable attention to the lifecycle management of software systems. Software is an integral part of many critical and non-critical applications, and virtually any industry- automotive, avionics, oil, telecommunications, banking, semiconductors etc., is dependent on computers for their basic functioning. As computer software permeates our modern society, and will continue to do so at an increasing pace in the future, software quality assurance becomes an issue of critical concern.

Software reliability is accepted as a key

*Supported by a contract from Charles Stark Draper Laboratory and in part by Bellcore as a core project in the Center for Advanced Computing and Communication

†Supported in part by National Science Foundation under grant number EEC-9418765.

factor in software quality since it quantifies software failures - which can make a powerful system inoperative[Lyu96]. It is defined as the probability of failure-free software operation for a specified period of time in a specified environment[MIO87] and is inversely related to the number of unrepaired faults in the system. Large software systems contain millions of lines of code and the sheer size of the product poses considerable problems in terms of the ability of software designers to rapidly achieve product quality. Most software errors are latent, i.e., they exist in the software system for a long time before they are detected, and they may remain unrepaired for a long time even after they are detected, which amplifies their impact.

Conventional software reliability models assume that the time to repair a fault is negligible and that the process of repairing a fault is perfect. This assumption is clearly impractical and needs to be amended in order to present more realistic software testing scenarios. The time of removal 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 software reliability models because it significantly complicates the failure process, making it impossible to obtain closed form expressions for various metrics of interest. The number of faults detected and removed by a particular time will depend on the actual time taken to remove the defect, and this number will be less than the instantaneous removal case. The residual number of faults in the software before it is deployed in the field are the soul cause of software unreliability, and hence is an extremely important measure for the software developer. This is specially true for the developer of a commercial off-the-shelf software package that will run on thousands of individual systems. The reliability of a commercial software is important to its users, however, they never report their reliability experience. They report the occurrence of a specific failure to the software development organization, with the presumption of getting the underlying fault fixed, so that the failure does not recur. Thus commercial software organizations focus on the residual number of faults, rather than reliability as a measure of software quality[Ken93]. Fault removal processes affect the residual number of faults in the software, and thus have a direct impact on the quality of a software product.

In this paper, we examine the various kinds of fault removal scenarios and analyze the effect of these various types of fault removal policies on the residual number of faults at the end of the testing process, using a non-homogeneous continuous time Markov chain. The fault removal rate is initially hypothesized to be constant, and is subsequently extended to cover cases covering time and state dependencies. These fault removal scenarios can be easily incorporated using the state space view of the non-homogeneous Poisson process.

The sequel of the paper is organized as follows. Section 2 presents an overview of finite failure non-homogeneous Poisson process (NHPP) models; Section 3 presents the state-space view of the non-homogeneous Poisson process software reliability models; Section 4 incorporates finite repair time into the model and presents various fault removal policies, Section 5 presents some numerical results, and Section 6 presents conclusions and future work.

2 Finite Failure NHPP Models

This section provides an overview of the finite failure non-homogeneous Poisson process software reliability models. This class of models is concerned with the number of faults detected in a given time and hence are also referred to as "fault count models"[GO79].

Software failures are assumed 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.

Given $\lambda(t)$, the mean value function $m(t) = E[N(t)]$, where $m(t)$ denotes the expected number of faults detected by time t, satisfies the relation,

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

and,

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

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

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

The time domain models which assume the failure process to be an NHPP differ in the approach they use for determining $\lambda(t)$ or $m(t)$. The NHPP models can be further classified into finite failures and infinite failures models.

Finite failures NHPP models assume that the expected number of failures observed during an infinite amount of testing time, or unlimited resources will be a finite number a[Far96]. Some of the popular finite failure NHPP models are discussed in the subsequent sections.

2.1 Goel-Okumoto Model

The Goel-Okumoto model is one of the most influential NHPP-based software reliability models. Its mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, are given by[GO79]

$$m(t) = a(1 - e^{-gt}) \tag{4}$$

and

$$\lambda(t) = age^{-gt} \tag{5}$$

where g is the failure occurrence rate per fault.

2.2 Generalized Goel-Okumoto Model

The GO model assumes that the failure intensity of the software system decreases as testing progresses. However, initially the testing team is not familiar with the software, hence the fault removal is slow, but after a certain time the team gains sufficient experience and knowledge about the behavior of a product under test which leads to higher rates of fault removal until a time is reached when a large number of faults have been detected and removed, thus becoming increasingly more difficult to detect and remove new ones. Therefore, the failure rate increases initially and then decreases. A variation of the GO model, known as the Generalized GO model[Goe85] was proposed to capture this increasing/decreasing failure rate. The mean value function, $m(t)$, and the failure intensity, $\lambda(t)$, of the software are given by

$$m(t) = a(1 - e^{-gt^{\gamma}}) \tag{6}$$

and

$$\lambda(t) = ag\gamma e^{-gt^{\gamma}} t^{\gamma - 1} \tag{7}$$

where g and γ reflect the quality of testing.

2.3 Delayed S-Shaped Model

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

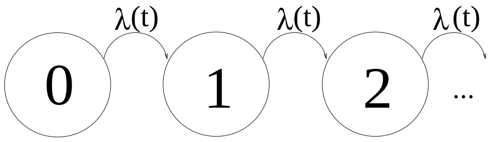

Figure 1. Non-homogeneous Markov chain for NHPP models

$m(t)$, and the failure intensity, $\lambda(t)$, in this case are given by[YOO83]

$$m(t) = a[1 - (1 + gt)e^{-gt}] \tag{8}$$

and

$$\lambda(t) = ag^2te^{-gt} \tag{9}$$

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

3 State-space view of NHPP

The NHPP models described in the previous section provide a closed-form analytical expression for the expected number of faults or mean value function, $m(t)$. However, the non-homogeneous Poisson process can also be represented by a non-homogeneous continuous time Markov chain as shown in Figure 1[Tri82].

The expected number of faults can also be computed numerically by solving the Markov chain shown in Figure 1 using SHARPE[ST87]. SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) is a software tool that analyzes stochastic models. For more information about the tool, the reader is referred to [STP95]. The chain in Figure 1 has infinite number of states, but for practical reasons the chain can be truncated to θ states, where θ is given by

$$\theta = \lceil a \rceil \tag{10}$$

where a is the expected number of failures that would be observed given infinite testing time or unlimited resources, as in case of the finite failure NHPP models.

The maximum likelihood estimate of a is obtained from the observed software failure data using CASRE[LN92]. SHARPE is designed to solve homogeneous CTMCs, but we get around this problem by dividing the time axis uniformly into small time intervals, where within each time interval, the failure intensity, $\lambda(t)$, can be assumed to be constant. Thus, within each time interval, the non-homogeneous continuous time Markov chain reduces to a homogeneous continuous time Markov chain which can then be solved using SHARPE. 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 then form the initial probability vector for the next time interval. Let $p_i(t)$ denote the probability of being in state i at time t. The mean value function, $m(t)$, can then be computed as

$$m(t) = \sum_{i=0}^{\theta} i * p_i(t) \tag{11}$$

The NTDS data[GO79, JM72] from the U.S. Navy Fleet Computer Programming Center consisting of errors in the development of software for a real-time, multicomputer complex which forms the core of the Naval Tactical Data System (NTDS) is used in this study. The NTDS software consisted of 38 different modules. Each module was supposed to follow three stages: the production (development) phase, the test phase, and the user phase. The parameters of the three NHPP models described above were estimated, and then the mean value function is computed for each of these models by solving the Markov chain in Figure 1. Figures 2, 3, and 4 show the analytical mean value function and the one obtained using a numerical solution of the non-homogeneous continuous time Markov chain (NHCTMC) for GO, S-

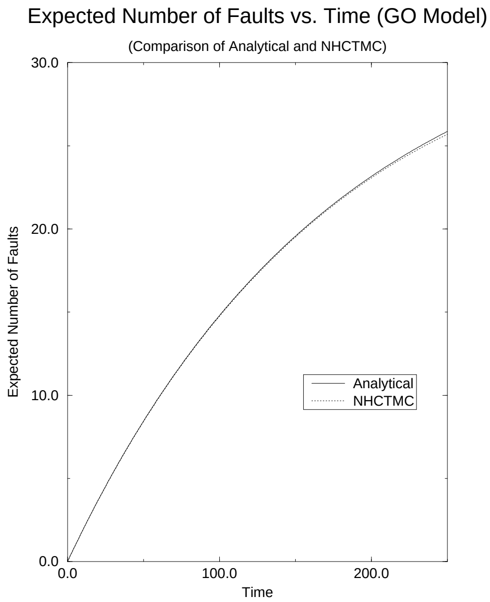

Figure 2. Analytical and numerical mean value functions - GO model

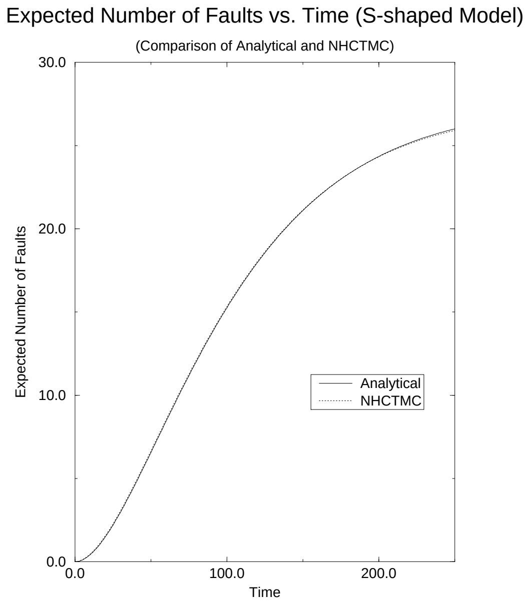

Figure 3. Analytical and numerical mean value functions - S-shaped model

shaped and the Generalized GO Models respectively.

As observed from Figures 2 - 4, the numerical solution of the mean value function obtained using the state-space view of the non-homogeneous Poisson process gives us a very good approximation to the analytical solution. This view enables us to incorporate more realistic features into the NHPP based software reliability models, which were initially based on oversimplifying assumptions in order to ensure mathematical tractability, as discussed in the subsequent sections.

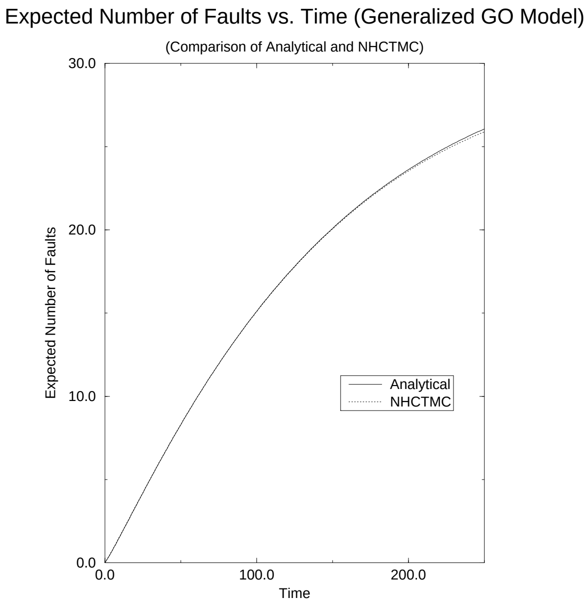

Figure 4. Analytical and numerical mean value functions - Generalized GO model

4 Fault Removal Process

The NHPP model presented as a non-homogeneous continuous time Markov chain (NHCTMC) is extended in this section to incorporate explicit fault removal process. Without loss of generality, we use the failure rate of the GO model. We assume that testing continues even during the repair process, and none of the faults are so severe that testing is rendered impossible. Thus, the faults are queued until they are repaired. The faults are repaired one at a time, and thus the detected faults form a queue up to a maximum of $\theta - l$, where l is the number of faults removed and θ is as given in Equation (10). The state space for the Markov chain in this case is a tuple (i, j), where i is the number of faults removed and j is the number of faults detected, pending to be removed.

The fault removal rate is assumed to be of the following types:

- Constant: This perhaps is the simplest possible situation where the fault removal rate is a constant denoted by μ, and the mean fault removal time is given by $1/\mu$. Figure 5 shows the non-homogeneous continuous time Markov chain (NHCTMC) with constant fault removal rate. A few attempts[Kre83, Lev90] made to incorporate explicit fault removal into the software reliability models have been restricted to this type.

- Fault dependent: The fault removal rate could depend on the number of faults pending for removal, since the more the number of faults pending, the quicker they are removed. Figure 6 shows the model where the fault removal rate depends on the number of faults pending for removal. If j is the number of faults pending for removal, the removal rate μ is given by
$$\mu = j * k \quad (12)$$

- Time-dependent: The fault removal rate could also depend on time since latent faults are harder to remove. The intuition behind this is that the fault removal rate is lower at the beginning of the testing and increases as testing progresses or as the deadline for the delivery of the software approaches, and reaches a constant value beyond which it cannot

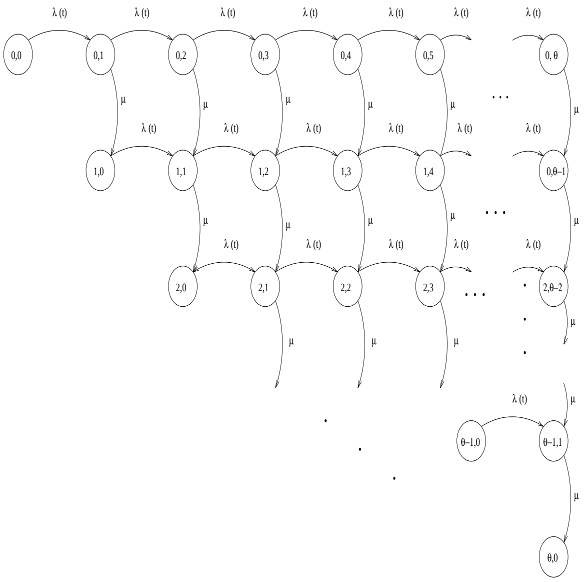

Figure 5. NHCTMC - Constant Fault Removal Rate

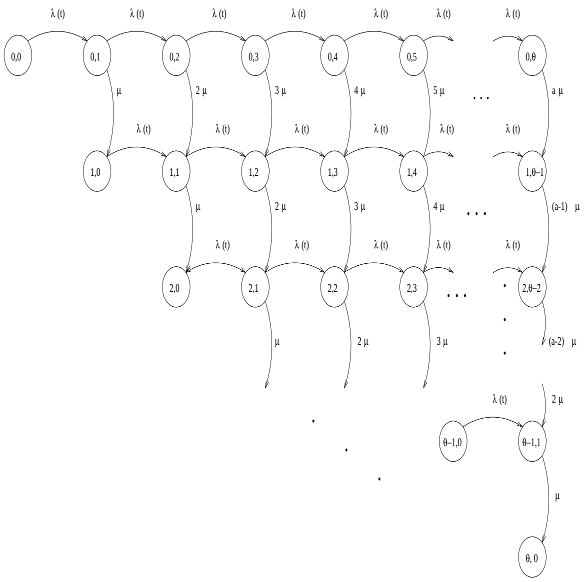

Figure 6. NHCTMC - Fault Dependent Removal Rate

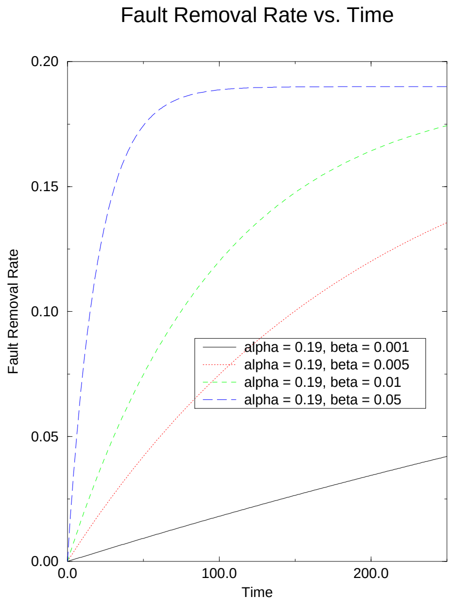

Figure 7. Time-dependent fault removal rate

increase. The time-dependent fault removal rate is hypothesized to be of the form

$$\mu(t) = \alpha(1 - e^{-\beta t}) \qquad (13)$$

Figure 7 shows the $\mu(t)$ for various values of α and β. In this case, the NHCTMC shown in Figure 5 is solved by approximating the time-dependent fault removal rate $\mu(t)$ at every time step in a manner similar to that of $\lambda(t)$, so that every time step we essentially solve a homogeneous Markov chain.

The expected number of faults removed, $m_R(t)$ and the expected number of faults detected, $m_D(t)$, by time t in case of Figure 5 and 6 is given by Equation (14), and (15) respectively.

$$m_R(t) = \sum_{i=0}^{\theta} \sum_{j=0}^{\theta-i} i * p_{i,j}(t) \qquad (14)$$

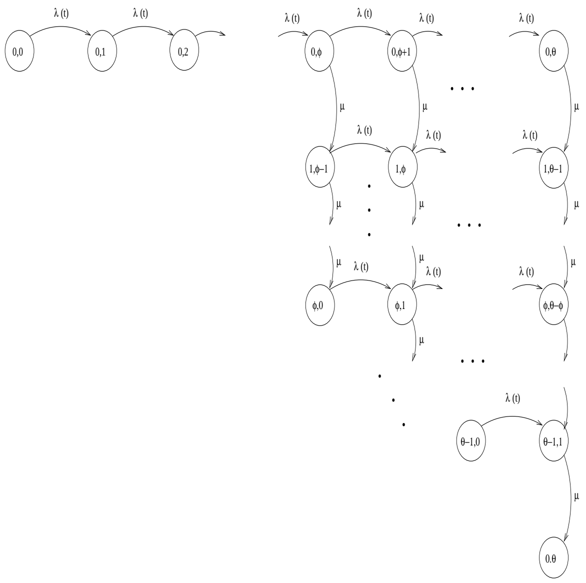

Figure 8. NHCTMC - Delayed (Fault Dependent Delay) Fault Removal Rate

$$m_D(t) = \sum_{i=0}^{\theta} \sum_{j=0}^{\theta-i} (i+j) * p_{i,j}(t) \qquad (15)$$

The process of fault removal can also be delayed in case of some software development projects. Delayed fault removal can be of two types:

- Fault removal can be deferred till a certain number ϕ of faults are detected and are pending to be removed. The NHCTMC in this case is as shown in Figure 8. The expected number of faults removed and detected are given by Equations (14) and (15) respectively.
- The fault removal process can be delayed and this delay can be incorporated into the NHCTMC using a phase type distribution[Tri82] as shown in Figure 9.

$1/\mu_1$ denotes the mean time in phase 1 and $1/\mu_2$ denotes the mean time in phase 2. The mean repair time $1/\mu$ is given by

$$\frac{1}{\mu} = \frac{1}{\mu_1} + \frac{1}{\mu_2} \tag{16}$$

In Figure 9, state (i, j, d) implies that i faults have been removed, j faults have been detected and are queued for removal, and "d" implies intermediate phase of repair. The expected number of faults removed, $m_R(t)$ and the expected number of faults detected, $m_D(t)$ is given by Equation (17) and (18) respectively.

$$m_R(t) = \sum_{i=0}^{\theta} \sum_{j=0}^{\theta-i} i * (p_{i,j}(t) + p_{i,j,d}(t)) \tag{17}$$

$$m_D(t) = \sum_{i=0}^{\theta} \sum_{j=0}^{\theta-i} (i+j) * (p_{i,j}(t) + p_{i,j,d}(t)) \tag{18}$$

5 Numerical Results

The non-homogeneous continuous time Markov chain (NHCTMC) with constant fault removal rate is solved for different values of μ, using SHARPE. Figure 10 shows the mean value function obtained by solving the NHCTMC in Figure 5 for various values of μ.

The expected number of faults removed decreases as the fault removal rate μ decreases which is quite expected. The cumulative defect removal curve is similar to the cumulative defect detection curve, and as the defect removal rate increases, the defect removal almost follows the defect detection curve, and we move closer and closer to the original assumption of instantaneous repair. However, as the fault removal rate is

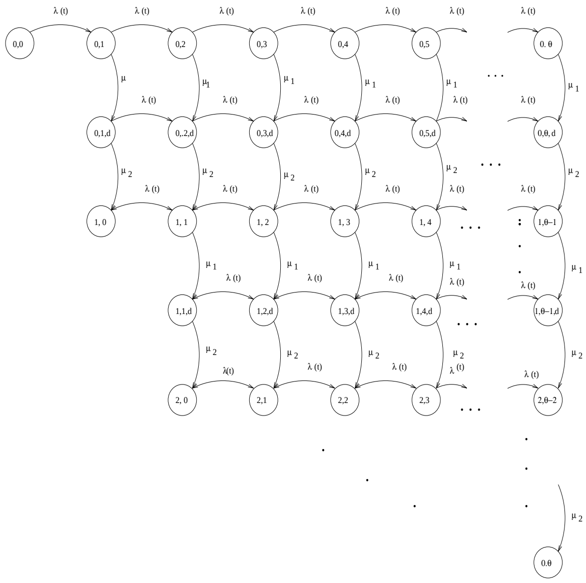

Figure 9. NHCTMC - (Time - Delayed) Fault Removal Rate

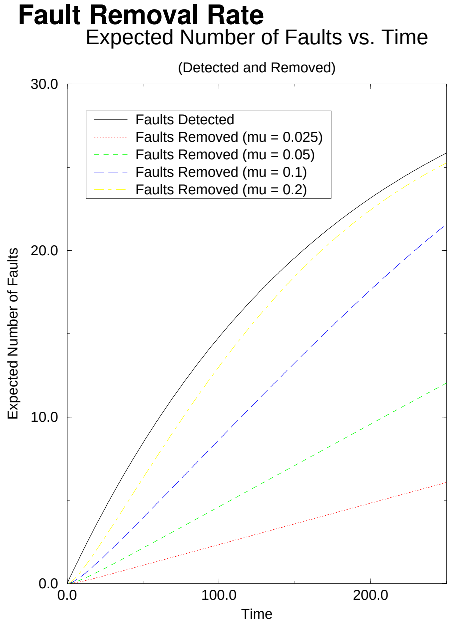

Figure 10. Mean Value Function - Constant Fault Removal Rate

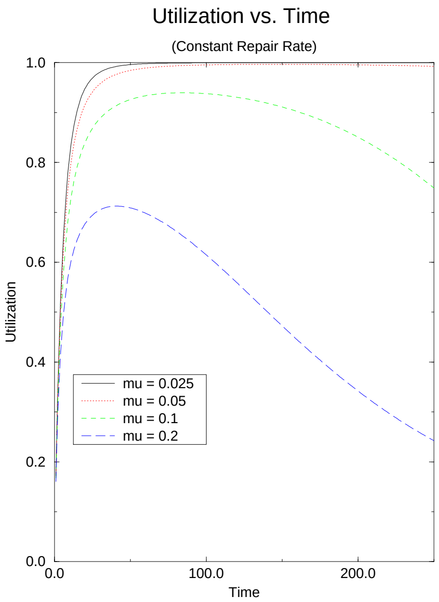

Figure 11. Utilization for Constant Fault Removal Rate

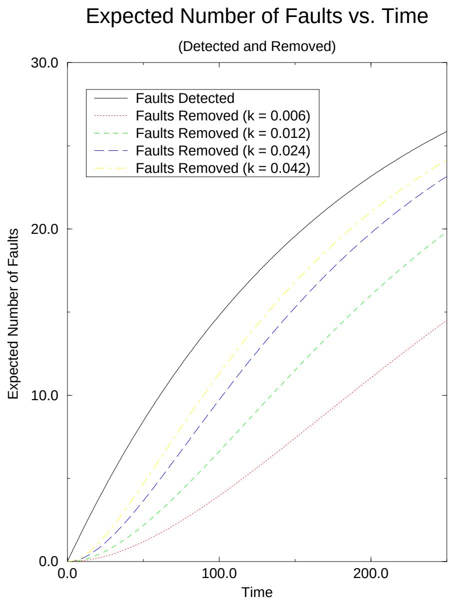

Figure 12. Mean Value Function - Fault Dependent Removal Rate

made higher and higher, there are increased chances that the fault removal mechanism remains idle during the testing process due to the lack of pending faults. The extent to which the fault removal mechanism is busy is reported by computing the utilization as shown in Figure 11.

The utilization could be used to establish bounds on the fault removal rates, since in the case of most software development projects, the debuggers are the same as software developers, and for cost-effective testing we would like to minimize the idle time as much as possible, at the same time achieve a desired level of software quality, by removing a maximum number of faults at the end of testing.

Figure 12 shows the expected number of faults removed as a function of time, for various values of k in Equation (12). The fault removal rate in this case depends on the number of pending faults, and the expected number of faults removed is directly related to the proportionality constant k i.e., the expected number of faults removed increases as k increases.

The expected fault removal rate in this case does not have a closed form expression and can be computed as a function of time, while solving the NHCTMC, and is shown in Figure 13.

The expected number of faults as a function of time for delayed fault removal where fault removal starts only after a certain number ϕ of faults is accumulated, is shown in Figure 14 for various values of ϕ. The cumulative defect removal curve is similar to the cumulative defect detection curve, except that it is skewed in time due to the defect repair delay. The actual delay depends upon the value of ϕ.

The expected number of faults as a function of time for a two phase-delayed repair,

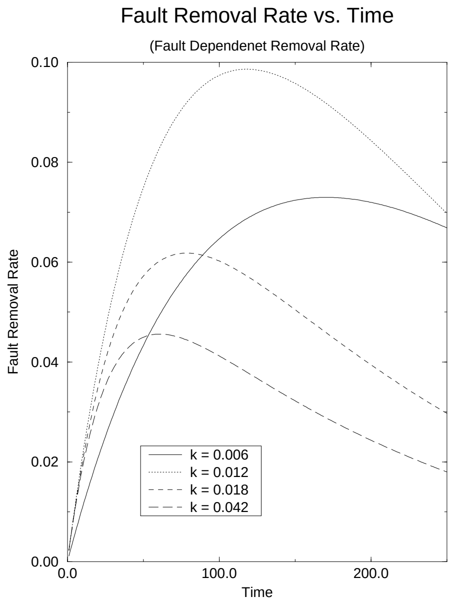

Figure 13. Fault Dependent Removal Rate

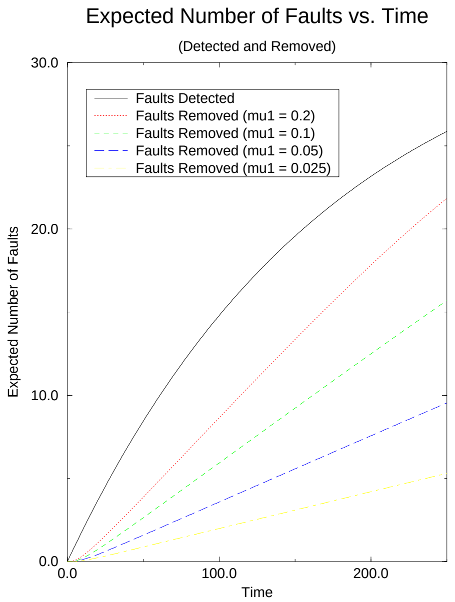

Figure 15. Mean Value Function - Delayed (Time) Removal

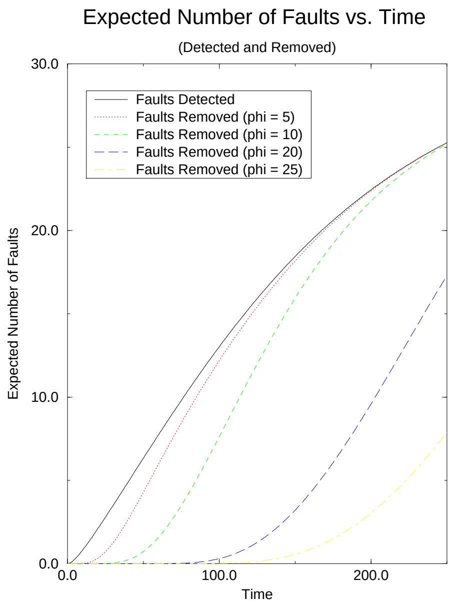

Figure 14. Mean value function - Delayed (Fault Dependent) Removal

for different values of μ_1, holding μ_2 constant at 0.2 is as shown in the Figure 15. The cumulative fault removal curve in this case is linear with respect to time, and the expected number of faults removed decreases as μ_1 decreases.

The expected number of faults in case of time-dependent fault removal rate is as shown in Figure 16. The cumulative defect removal curve in this case also is similar to the defect detection curve, except that it is delayed in time, and this delay depends on the value of β, since α is held constant, where α and β are as per Equation (13).

6 Conclusions and Future Work

In this paper, we have incorporated explicit fault removal into the finite failure NHPP models, which assume instantaneous repair to ensure mathematical tractability. Using the state-space view of the non-homogeneous Poisson process, the unrealistic assumption of immediate repair can be re-

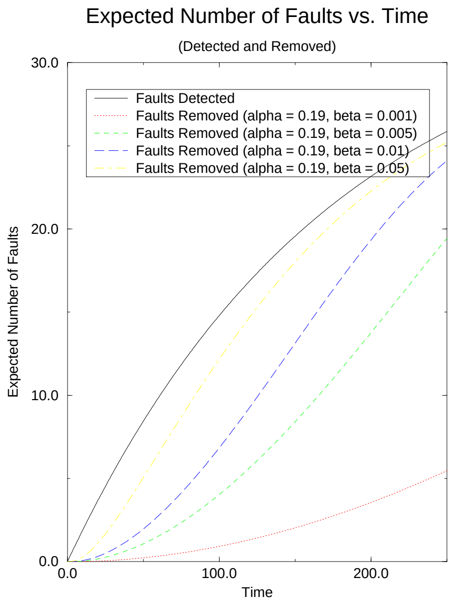

Figure 16. Mean Value Function - Time Dependent Fault Removal

laxed, however, we have to rely on the numerical solution of the Markov chain, rather than obtaining a closed-form expression for the mean value function. Various types of fault removal policies have been studied, viz., constant fault removal rate, time dependent fault removal rate, and delayed repair. In general, finite fault removal time, reduces the number of faults removed at the end of testing time, or increases the residual number of faults in the software at the end of testing, and thus the estimate of the quality of the software product obtained using the NHCTMC model with explicit fault removal will be more realistic than that obtained from models using instantaneous repair.

The NHCTMC can be extended to incorporate fault reintroduction during removal process, along with the various repair policies. Predictions in the operational phase can be made using the NHCTMC, and stopping rules can be developed for optimum software release times.

The NHCTMC model should be validated using data from real software development efforts. Simulation techniques can be explored for a more complicated fault removal process.

7 Acknowledgments

The authors wish to acknowledge Dr. J. Robert Horgan of Bell Communications Research and Dr. Amrit Goel of Syracuse University for their valued input.

References

[Far96] W. Farr. *Handbook of Software Reliability Engineering, M. R. Lyu, Editor*, chapter Software Reliability Modeling Survey, pages 71–117. McGraw-Hill, New York, NY, 1996.

[GO79] A. L. Goel and K. Okumoto. "Time-Dependent Error-Detection Rate Models for Software Reliability and Other Performance measures". *IEEE Trans. on Reliability*, R-28(3):206–211, August 1979.

[Goe85] A. L. Goel. "Software Reliability Models: Assumptions, Limitations and Applicability". *IEEE Trans. on Software Engineering*, SE-11(12):1411–1423, December 1985.

[JM72] Z. Jelinski and P. B. Moranda. *Statistical Computer Performance Evaluation, ed. W. Freiberger*, chapter Software Reliability Research, pages 465–484. Academic Press, New York, 1972.

[Ken93] G. Q. Kenney. "Estimating Defects in Commercial Software During Operational Use". *IEEE Trans. on Reliability*, 42(1):107–115, January 1993.

[Kre83] W. Kremer. "Birth and Death Bug Counting". *IEEE Trans. on Reliaability*, R-32(1):37–47, April 1983.

[Lev90] Y. Levendel. "Reliability Analysis of Large Software Systems: Defect Data Modeling". *IEEE Trans. on Software Engineering*, 16(2):141–152, February 1990.

[LN92] M. R. Lyu and A. P. Nikora. "CASRE-A Computer-Aided Software Reliability Estimation Tool". In *CASE '92 Proceedings*, pages 264–275, Montreal, Canada, July 1992.

[Lyu96] M. R. Lyu. *Handbook of Software Reliability Engineering, M. R. Lyu, Editor*, chapter Introduction, pages 3–25. McGraw-Hill, New York, NY, 1996.

[MIO87] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability - Measurement, Prediction, Application.* McGraw Hill, New York, 1987.

[ST87] R. A. Sahner and K. S. Trivedi. "Reliability Modeling Using SHARPE". *IEEE Trans. on Reliability*, R-36(2):186–192, June 1987.

[STP95] 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.

[Tri82] K. S. Trivedi. *"Probability and Statistics with Reliability, Queuing and Computer Science Applications"*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[YOO83] S. Yamada, M. Ohba, and S. Osaki. "S-Shaped Reliability Growth Modeling for Software Error Detection". *IEEE Trans. on Reliability*, R-32(5):475–485, December 1983.