International Journal of Reliability and Applications
Vol. 7, No. 2, pp. 155-166, 2006

Frameworks for NHPP Software Reliability Growth Models

J.-Y. Park
Department of Information Statistics, Gyeongsang National University
900 Gazwa-dong Jinju, 660-701, Republic of Korea

J.-H. Park
Department of Computer Science, Gyeongsang National University
900 Gazwa-dong Jinju, 660-701, Republic of Korea

T. Fujiwara
Development Dept. 2, Development Division, Fujitsu Peripherals Limited
35 Saho, Katoh-shi, Hyogo, 673-1447, Japan

Abstract. Many software reliability growth models (SRGMs) based on nonhomogeneous Poisson process (NHPP) have been developed and applied in practice. NHPP SRGMs are characterized by their mean value functions. Mean value functions are usually derived from differential equations representing the fault detection/removal process during testing. In this paper such differential equations are regarded as frameworks for generating mean value functions. Currently available frameworks are theoretically discussed with respect to capability of representing the fault detection/removal process. Then two general frameworks are proposed.

Key Words : *construct, coverage growth function, fault detection rate, fault introduction rate, fault removal rate, framework, mean value function, repeated construct execution, software reliability growth model.*

1. INTRODUCTION

Many SRGMs have been proposed to estimate the reliability of a software system and the number of faults remaining in the software system. Among all SRGMs, NHPP SRGMs have been widely and successfully applied in practical software reliability engineering. NHPP SRGMs assume that the number of faults detected up

Corresponding Author.
E-mail address: joongyang@gnu.ac.kr

to testing time t follows an NHPP with mean value function (MVF) $m(t)$. NHPP SRGMs are therefore characterized by their MVFs. Capability of an NHPP SRGM depends on whether its MVF represents well the fault detection/removal process during testing. A number of differential equations for MVF have been proposed. Such a differential equation is able to generate differenent MVFs. We thus regard the differential equations as frameworks for NHPP SRGMs. In this paper we discuss the currently available frameworks with respect to the capability of representing the fault detection/removal process. The available frameworks can be classified into two classes, time-domain frameworks and coverage-based frameworks. Time-domain frameworks depend on the testing time, whereas coverage-based frameworks depend on the coverage growth behavior. Time-domain frameworks are considered in Section 2. Coverage-based frameworks are discussed in Section 3. General coverage-based frameworks are then introduced in Sections 4 and 5. Finally Section 6 presents concluding remarks.

2. TIME-DOMAIN FRAMEWORKS

The earliest time-domain framework, proposed by Goel and Okumoto (1978) and henceforth called F_1, is given as

$$\frac{dm(t)}{dt} = b(t)\,[a - m(t)]\,, \tag{2.1}$$

where a is the number of initial faults. Framework F_1 has been derived from the following assumptions:

(A1) The debugging is perfect, i.e., each detected fault is removed without introducing new faults.

(A2) The fault detection rate (FDR) of a software system at any time is proportional to the number of faults remaining in the software system. The proportionality $b(t)$ is called the FDR per remaining fault at time t.

If an appropriate FDR per remaining fault is chosen and incorporated into F_1, a specific MVF is obtained. That is, F_1 generates different MVFs for different FDRs.

In reality, debugging a detected fault is not always successful. The detected fault may not be removed and even some new faults may be introduced into the software during debugging. It is therefore desirable to modify F_1 so that the imperfect debugging is taken into account. Pham and Nordmann (1997) generalized F_1 by replacing (A1) with

(A1′) The debugging is imperfect.

The generalization of F_1 is called F_2 and given as

$$\frac{dm(t)}{dt} = b(t)\,[a(t) - m(t)]\,, \tag{2.2}$$

where $a(t)$ is the fault content function. The fault content function represents the number of initial faults and faults introduced by the imperfect debugging up to time t. The imperfect debugging is therefore reflected in the fault content function. However, F_2 has a critical problem that $a(t) - m(t)$ is not the number of remaining faults, but the number of undetected faults. This is because $m(t)$ is the number of detected faults and the number of detected faults is not generally equal to the number of removed faults in the imperfect debugging environment. The imperfect debugging has two aspects. One is the imperfect fault removal. The other is the fault introduction. Framework F_2 does not consider the imperfect fault removal. Consequently, F_2 does not represent (A2) properly under (A1′).

Zhang et al. (2003) have modified F_2 so that (A2) is properly reflected in the imperfect debugging environment. They itemize (A1′) into two assumptions, which describe the two aspects of the imperfect debugging.

(A3) A detected fault is immediately debugged and successfully removed with fault removal rate (FRR) $b_r(t)$.

(A4) Whether a detected fault is successfully removed or not, some new faults may be introduced into the software with fault introduction rate (FIR) $b_i(t)$.

Note $b_i(t) = 0$ and $b_r(t) = 1$ meaning the perfect debugging. Zhang et al. (2003) performed the modification assuming that FRR is constant, i.e., $b_r(t) = b_r$. The modified framework named F_3 is derived from (A2)-(A4) and represented by the following differential equations:

$$\frac{dm(t)}{dt} = b(t)\,[a(t) - b_r\, m(t)]\,, \tag{2.3}$$

$$\frac{da(t)}{dt} = b_i(t)\,\frac{dm(t)}{dt}\,. \tag{2.4}$$

NHPP SRGMs including some well-known NHPP SRGMs can be derived by incorporating combinations of FDR, FIR, FRR and the fault content function into the time-domain frameworks. Performance of the NHPP SRGMs has been studied and compared by Pham and Zhang (1997), Zhang and Pham (2000), Pham (2003) and Zhang et al. (2003). Such performance analysis is equivalent to performance comparison of combinations of FDR, FIR, FRR and the fault content function under the assumption that the corresponding time-domain frameworks are reasonable. If the time-domain frameworks do not represent the fault detection/removal process well, such performance analysis would be meaningless. A fundamental point is therefore whether the time-domain frameworks reasonably model the fault detection/removal process.

A common drawback of the time-domain frameworks is associated with (A2). Assumption (A2) means that all the remaining faults are detectable at any testing time. However, this does not hold in general. When a test case is run, only a portion of the software system is executed and only the faults in the executed portion

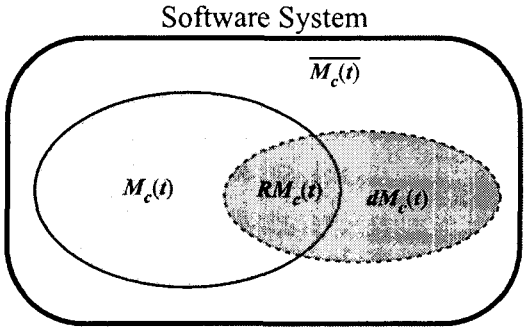

Figure 1. Relationships among $M_c(t)$, $\overline{M_c(t)}$, $dM_c(t)$ and $RM_c(t)$.

are detectable in the fault detection activity. This implies that only a portion of the remaining faults is detectable and that (A2) is not realistic. Consequently, frameworks based on (A2) are not reasonable. Frameworks are required to be based on the detectable faults rather than the remaining faults. Such frameworks will be considered in the subsequent sections.

3. COVERAGE-BASED FRAMEWORKS

We begin this section by defining terms and notations necessary for describing the coverage growth. A building block of a software is called a construct. It depends on the coverage metric. If the statement coverage is measured, a construct is a statement. If the block coverage is measured, a construct is a block. We denote by M the set of all the constructs of a software system. The number of covered constructs increases along with the testing progress. We represent the set of constructs covered up to testing time t by $M_c(t)$. The coverage at testing time t is computed as $C(t) = |M_c(t)||M|^{-1}$, where $|\cdot|$ be the cardinality of a set of constructs. Since the number of constructs executed by a test case is usually modeled as a random variable, $|M_c(t)|$ and $C(t)$ are also considered as random variables. Thus the coverage growth function (CGF) is defined as the expected value of $C(t)$, i.e., $c(t) = E[C(t)]$. It should be noted that generally the constructs in $M_c(t)$ may be executed repeatedly. This phenomenon is called the repeated construct execution. Suppose that additional testing is performed during $(t, t+dt]$. The additional testing will then execute some constructs in $\overline{M_c(t)} = M - M_c(t)$ and some constructs in $M_c(t)$. The constructs newly covered during $(t, t+dt]$ constitute the increment of $M_c(t)$ and $dM_c(t)$. The increment of $c(t)$ corresponding to $dM_c(t)$ is denoted by $dc(t)$. The set of constructs re-executed during $(t, t+dt]$ is denoted by $RM_c(t)$. Relationships among $M_c(t)$, $\overline{M_c(t)}$, $dM_c(t)$ and $RM_c(t)$ are shown in Figure 1. Since only the faults in $dM_c(t)$ and $RM_c(t)$ are exposed to the fault detection activity, they are the faults which are detectable during $(t, t+dt]$.

Piwowarski et al. (1993) proposed a coverage-based framework, which is represented as

$$\frac{dm(t)}{dt} = a\frac{dc(t)}{dt} = \frac{a}{|M|}\frac{|dM_c(t)|}{dt}. \tag{3.1}$$

This framework, referred to as F_4, assumes that only the faults in $dM_c(t)$ are detectable and all the detectable faults are detected. The number of faults in $dM_c(t)$ can be computed as the product of the fault density per construct in $dM_c(t)$ and the number of constructs in $dM_c(t)$. The fault density per construct in $dM_c(t)$ is equal to the fault density per construct in $\overline{M_c(t)}$. The fault density per construct in $\overline{M_c(t)}$ is equal to the initial fault density per construct at the beginning of testing. Framework F_4 computes the fault density per construct in $dM_c(t)$ as $a|M|^{-1}$. This computation is valid when the initial faults are uniformly distributed over M. Since F_4 does not take the faults in $RM_c(t)$ into account, F_4 is valid when there are no faults in $RM_c(t)$. This holds if all the detectable faults are detected and removed. Therefore, F_4 is based on (A1) and the following assumptions:

(A5) The number of faults detected during $(t, t+dt]$ is proportional to the number of faults in $dM_c(t)$. The proportionality $b_d(t)$ is called the FDR per detectable fault.

(A6) The fault detection is perfect, i.e., $b_d(t) = 1$.

(A7) All the initial faults are uniformly distributed over M.

Application of F_4 is limited to the case where the fault detection and the fault debugging are perfect. Applicability of F_4 can be extended by relieving the perfect fault detection requirement and/or the perfect fault debugging requirement. We now discuss three coverage-based frameworks, which can be regarded as generalizations of F_4 in some respects.

$$\frac{dm(t)}{dt} = b_d(t)\,a\frac{dc(t)}{dt} = b_d(t)\frac{a}{|M|}\frac{|dM_c(t)|}{dt}, \tag{3.2}$$

$$\frac{dm(t)}{dt} = [a(t) - m(t)]\frac{\frac{dc(t)}{dt}}{1-c(t)} = \frac{a(t)-m(t)}{|\overline{M_c(t)}|}\frac{|dM_c(t)|}{dt}, \tag{3.3}$$

$$\frac{dm(t)}{dt} = b_d(t)\,[a - m(t)]\frac{dc(t)}{dt} = b_d(t)\frac{a-m(t)}{|M|}\frac{|dM_c(t)|}{dt}. \tag{3.4}$$

These are respectively called F_5, F_6 and F_7. Framework F_5 has been proposed by Gokhale et al. (1996) and further studied by Gokhale and Trivedi (1999). Pham and Zhang (2003) have suggested F_6 as a specialization of F_2 for $b(t) = \frac{dc(t)}{dt}/[1-c(t)]$. Yamamoto et al. (2004) have proposed F_7 as a specialization of F_1 for $b(t) = b_d(t)\frac{dc(t)}{dt}$. In fact, Yamamoto et al. (2004) have considered the case where $b_d(t)$ is

constant. As in F_4, the right hand side of these frameworks is the product of three terms: FDR per detectable fault, the fault density per construct in $dM_c(t)$ and the number of constructs in $dM_c(t)$. It should be noted that the FDR per detectable fault is 1 in both F_4 and F_6 and that the fault content function is involved in F_6. Since the product of the fault density per construct in $dM_c(t)$ and the number of constructs in $dM_c(t)$ is the number of faults in $dM_c(t)$, all the three frameworks assume that only the faults remaining in $dM_c(t)$ are detectable. That is, (A5) is a common assumption for F_5-F_7. However, F_5-F_7 compute the fault density per construct in $dM_c(t)$ in different ways. We now present additional assumptions necessary for F_5-F_7.

(A6′) The fault detection is imperfect, i.e., $0 < b_d(t) < 1$.

(A7′) All the undetected (not remaining) faults are uniformly distributed over $\overline{M_c(t)}$.

(A7″) All the remaining faults are uniformly distributed over M.

Frameworks F_5 and F_7 have been developed under the perfect debugging assumption (A1), whereas F_6 has been developed under the imperfect debugging assumption (A1′). Frameworks F_5-F_7 have been developed under the imperfect fault detection assumption (A6′). Nonetheless, F_6 does not reflect (A6′). Therefore, F_5 is based on (A1), (A5), (A6′) and (A7); F_6 is based on (A1′), (A5), (A6) and (A7′); F_7 is based on (A1), (A5), (A6′) and (A7″). In discussing F_4, we concluded that (A5) is valid when both (A1) and (A6) hold. If the debugging is imperfect, the detected faults may not be removed from $M_c(t)$. Even some new faults may be introduced into the software. The introduced faults are usually located in $M_c(t)$. If the fault detection is imperfect, $M_c(t)$ possibly contains the undetected faults. Therefore, when one or both of (A1) and (A6) is violated, the faults in $RM_c(t)$ should be included in the set of detectable faults. Each of F_5-F_7 violates either (A1) or (A6). However, F_5-F_7 do not take the faults in $RM_c(t)$ into account. This is the common drawback of F_5-F_7. Furthermore, (A7′) and (A7″) are not reasonable for computing the fault density per construct in $dM_c(t)$. The undetected detectable faults and the introduced faults are in $M_c(t)$. The constructs in $\overline{M_c(t)}$ and $dM_c(t)$ are the same as they were at the beginning of testing. Assumptions (A7′) and (A7″) are equivalent to (A7) at the beginning of testing. Therefore, the fault density per construct in $dM_c(t)$ is $a\,|M|^{-1}$.

The four coverage-based frameworks F_4-F_7 have been proposed for the general testing strategy. There are two frameworks, which have been developed for the resource-constrained nonoperational testing. The resource-constrained nonoperational testing generates test cases so as to cover as many uncovered constructs as possible and ignores $RM_c(t)$. Therefore, only the faults in $dM_c(t)$ are detectable and (A5) is valid for the resource-constrained nonoperational testing. Let $m(t) = \tilde{m}\,(c(t))$ and $b_d(t) = \tilde{b}_d\,(c(t))$. That is, $\tilde{m}(c)$ and $\tilde{b}_d(c)$ are respectively the MVF and FDR per detectable fault expressed in coverage. Two coverage-based

frameworks involving $\tilde{m}(c)$ and $\tilde{b}_d(c)$ have been suggested by Vouk (1992) and Rivers and Vouk (1995). The two frameworks, referred to as F_8 and F_9, are represented by

$$\frac{d\tilde{m}(c)}{dc} = \tilde{b}_d(c)\ [a - \tilde{m}(c)]\ , \tag{3.5}$$

$$\frac{d\tilde{m}(c)}{a - \tilde{m}(c)} = \tilde{b}_d(c)\frac{dc}{1-c}\ . \tag{3.6}$$

In fact, Vouk (1992) has considered only the case where $\tilde{b}_d(c) = \beta\,(c - c_{min})$ and $c_{min} \leq c \leq 1$. Framework F_9 has been studied further by Rivers and Vouk (1998) and Vouk and Rivers (2003). Frameworks F_8 and F_9 can be transformed to frameworks in testing time as follows:

$$\frac{dm(t)}{dt} = b_d(t)\ [a - m(t)]\ \frac{dc(t)}{dt} = b_d(t)\,\frac{a - m(t)}{|M|}\,\frac{|dM_c(t)|}{dt}\ , \tag{3.7}$$

$$\frac{dm(t)}{dt} = b_d(t)\ [a - m(t)]\ \frac{\frac{dc(t)}{dt}}{1 - c(t)} = b_d(t)\,\frac{a - m(t)}{|\overline{M_c(t)}|}\,\frac{|dM_c(t)|}{dt}\ . \tag{3.8}$$

Framework F_8 is identical to F_7, whereas F_9 is very similar to F_6. Thus we can easily obtain the assumptions corresponding to F_8 and F_9. Assumptions for F_8 are (A1), (A5), (A6′) and (A7″), whereas those for F_9 are (A1), (A5), (A6′) and (A7′). Assumption (A5) is valid for the resource-constrained nonoperational testing. Both F_8 and F_9 fail to compute the fault density per construct in $dM_c(t)$. As in F_5-F_7, the fault density per construct in $dM_c(t)$ is $a\,|M|^{-1}$. If $a\,|M|^{-1}$ is used as the fault density per construct in $dM_c(t)$, F_8 and F_9 become identical to F_5. That is, F_5 is a valid framework for the resource-constrained nonoperational testing.

4. A GENERAL COVERAGE-BASED FRAMEWORK FOR PERFECT DEBUGGING

We have shown in the previous two sections that the currently available frameworks have shortcomings in some respects. We thus introduce a general coverage-based framework for the perfect debugging without such shortcomings. The general framework, denoted by F_{10}, is derived from (A1), (A6′), (A7) and

(A5′) The number of faults detected during $(t, t + dt]$ is proportional to the number of faults in $dM_c(t)$ and $RM_c(t)$. The proportionality $b_d(t)$ is called the FDR per detectable fault.

(A8) The faults remaining in $M_c(t)$ are uniformly distributed over $M_c(t)$.

Assumption (A5′) implies that the number of detectable faults is the faults in $dM_c(t)$ and $RM_c(t)$. Assumption (A8) is introduced for computing the fault density per

construct in $M_c(t)$. Generally the faults remaining in $M_c(t)$ are not uniformly distributed, even when the initial faults are uniformly distributed. Some constructs enter $M_c(t)$ in the early stage of testing. Some constructs enter $M_c(t)$ in the later stage of testing. The fault density of the constructs entering $M_c(t)$ early tends to be higher than that of the constructs entering $M_c(t)$ later. This is because the FDR per detectable fault usually increases in testing time. On the other hand, the constructs entering $M_c(t)$ early are likely to be re-executed more times than the constructs entering $M_c(t)$ later. That is, the constructs entering $M_c(t)$ with relatively high fault density are repeatedly exposed to the fault detection activity more frequently than the constructs entering $M_c(t)$ with relatively low fault density. Thus we assume that the fault densities of the constructs in $M_c(t)$ are approximately equal.

Let us denote the number of initial faults and the number of faults remaining in a set of constructs by $n_T(\cdot)$ and $n_R(\cdot)$ respectively. Then the increment of $m(t)$ during $(t, t+dt]$ is obtained as

$$dm(t) = b_d(t)\left[n_R\left(RM_c(t)\right) + n_T\left(dM_c(t)\right)\right]. \tag{4.1}$$

The initial fault density per construct is $a|M|^{-1}$ due to (A7). Therefore, $n_T\left(M_c(t)\right)$ and $n_T\left(dM_c(t)\right)$ are obtained as

$$n_T\left(M_c(t)\right) = \frac{a}{|M|}\,|M_c(t)| = a\,c(t), \tag{4.2}$$

and

$$n_T\left(dM_c(t)\right) = \frac{a}{|M|}\,|dM_c(t)| = a\,dc(t). \tag{4.3}$$

Since $n_R\left(M_c(t)\right) = n_T\left(M_c(t)\right) - m(t)$, $n_R\left(RM_c(t)\right)$ is computed from (A8) as

$$n_R\left(RM_c(t)\right) = \frac{n_R\left(M_c(t)\right)}{|M_c(t)|}\,|RM_c(t)| = \frac{ac(t) - m(t)}{c(t)}\,\frac{|RM_c(t)|}{|M|}. \tag{4.4}$$

Substituting Eqs. (4.3) and (4.4) into Eq. (4.1) and dividing both sides of Eq. (4.1) by dt, we obtain the following framework:

$$\begin{aligned}\frac{dm(t)}{dt} &= b_d(t)\left[\frac{ac(t) - m(t)}{c(t)}\,\frac{\frac{|RM_c(t)|}{|M|}}{dt} + a\,\frac{dc(t)}{dt}\right] \\ &= b_d(t)\left[\frac{ac(t) - m(t)}{c(t)}\left[\lambda - \frac{dc(t)}{dt}\right] + a\,\frac{dc(t)}{dt}\right],\end{aligned} \tag{4.5}$$

where λ is the expected proportion of constructs executed during a unit testing time.

Apparently, F_{10} given by Eq. (4.5) is characterized by the time-dependent behavior of $|RM_c(t)|$ and $c(t)$, equivalently λ and $c(t)$. Their behavior is determined by the testing strategy. For example, consider the resource-constrained nonoperational testing. The framework for the resource-constrained nonoperational testing

can be obtained by replacing $|RM_c(t)|$ in F_{10} with 0. Framework F_{10} is then simplified to F_5. This implies that F_5 is applicable to the resource-constrained non-operational testing, not to a general testing. Next consider the uniform testing in which all the constructs in M are equally likely executed. It can be shown that $|RM_c(t)|\,|dM_c(t)|^{-1} = c(t)[1-c(t)]^{-1}$ and $c(t) = 1-e^{-\lambda t}$ are for the uniform testing. Therefore, the general framework F_{10} reduces to

$$\frac{dm(t)}{dt} = \lambda\, b_d(t)\,[a - m(t)]\,. \tag{4.6}$$

Therefore, F_{10} for the uniform testing is equivalent to F_1. This implies that F_1 is applicable to the uniform testing, not to a general testing. As the last example, consider the case where the fault detection is perfect. Then there will be no faults in $M_c(t)$. That is, $b_d(t) = 1$ and $ac(t) - m(t) = 0$ are for the perfect debugging environment. Framework F_{10} then becomes F_4. This implies that F_4 is applicable to the perfect fault detection case.

5. A GENERAL COVERAGE-BASED FRAMEWORK FOR IMPERFECT DEBUGGING

In this section we extend the general framework developed in the previous section to the case where the debugging is imperfect. It will be derived from assumptions (A3), (A4), (A5′), (A6′), (A7) and (A8). Let $i(t)$ and $r(t)$ denote the number of introduced faults and the number of removed faults respectively. Since the number of detectable faults is $n_R\,(RM_c(t)) + n_T\,(dM_c(t))$, we obtain the following differential equations:

$$\frac{di(t)}{dt} = b_i(t)\,\frac{dm(t)}{dt}\,, \tag{5.1}$$

$$\frac{dr(t)}{dt} = b_r(t)\,\frac{dm(t)}{dt}\,, \tag{5.2}$$

$$\frac{dm(t)}{dt} = b_d(t)\,\frac{n_R\,(RM_c(t)) + n_T\,(dM_c(t))}{dt}\,. \tag{5.3}$$

Since $n_T\,(dM_c(t))$ is obtained as Eq. (4.3) and

$$n_R\,(RM_c(t)) = \frac{n_R\,(RM_c(t))}{|M_c(t)|}\,|RM_c(t)| = \frac{ac(t) - r(t) + i(t)}{c(t)}\,\frac{|RM_c(t)|}{|M|}\,, \tag{5.4}$$

Eq. (5.3) is written as

$$\begin{aligned}\frac{dm(t)}{dt} &= b_d(t)\left[\frac{ac(t) - r(t) + i(t)}{c(t)}\,\frac{\frac{|RM_c(t)|}{|M|}}{dt} + a\,\frac{dc(t)}{dt}\right] \\ &= b_d(t)\left[\frac{ac(t) - r(t) + i(t)}{c(t)}\left[\lambda - \frac{dc(t)}{dt}\right] + a\,\frac{dc(t)}{dt}\right]\,.\end{aligned} \tag{5.5}$$

The general framework for imperfect debugging, called F_{11}, is therefore represented by Eqs. (5.1), (5.2) and (5.5).

The main difference between F_{10} and F_{11} is computation of $n_R(RM_c(t))$. The difference results from whether the debugging is perfect or not. If the debugging is perfect, then $b_i(t) = 0$ and $b_r(t) = 1$. Consequently $r(t) = m(t)$ and $i(t) = 0$. Therefore, F_{11} becomes F_{10}. Applying $|RM_c(t)|\,|dM_c(t)|^{-1} = c(t)[1 - c(t)]^{-1}$ and $c(t) = 1 - e^{-\lambda t}$ to F_{11}, we can show that F_{11} for the uniform testing is obtained as

$$\frac{dm(t)}{dt} = \lambda\, b_d(t)\, [a - r(t) + i(t)]\,. \tag{5.6}$$

Since $[a - r(t) + i(t)]$ is the number of faults remaining in the software, Eq. (5.6) can be regarded as a version of F_1 for the imperfect debugging. Assuming that $b_r(t) - b_i(t)$ is constant, Park et al. (2005) derived NHPP SRGMs from Eq. (5.6) for various $b_d(t)$. If we let $a(t) = a + i(t)$ and $b_r(t) = b_r$, F_{11} reduces to F_3. This implies that F_3 is applicable to the case where the testing is performed according to the uniform testing profile and the fault removal rate is constant.

6. CONCLUDING REMARKS

In this paper we have compared the currently available frameworks for NHPP SRGMs. It has been shown that each of the available frameworks does not adequately represent the general fault detection/removal process in some respects. Specifically, the imperfect fault detection, the imperfect debugging and the repeated construct execution are not reflected successfully in the available frameworks. Thus, we have introduced general coverage-based frameworks that the imperfect fault detection, the imperfect debugging, and the repeated construct execution are explicitly incorporated. However, solutions to the general frameworks are not easily obtained except for the uniform testing. Solutions and practical application of the general frameworks will be studied in the future. CGF, FDR per detectable fault, FIR, and FRR are important components of both the available frameworks and the proposed general frameworks. In general, CGF depends on the testing strategy, whereas FDR, FIR, and FRR depend on the testers' skill and the testing environment. Insertion of different combinations of CGF, FDR per detectable fault, FIR, and FRR into the general frameworks generates different NHPP SRGMs. Performance of such an NHPP SRGM depends on how closely its CGF, FDR, FIR, and FRR represent the coverage growth behavior, the fault detection ability and the debugging ability, respectively. The available CGFs were studied theoretically and empirically by Park et al. (2005) and Park and Fujiwara (2006). It has been found that only two of the available CGFs are able to represent the coverage growth behavior. However, FDRs, FIRs, and FRRs have not been studied yet. Our future study will be directed to performance analysis of the available FDRs, FIRs, and FRRs and development of new FDRs, FIRs, and FRRs.

REFERENCES

A. L. Goel, and K. Okumoto (1978). A time dependent error detection rate model for a large scale software system. *Proceedings of 3rd USA-Japan Computer Conference*, 35-40.

S. S. Gokhale, T. Philip, P. N. Marinos, and K. S. Trivedi (1996). Unification of finite failure non-homogeneous Poisson process models through test coverage. *Proceedings of 7th IEEE International Symposium on Software Reliability Engineering*, White Plains, New York, USA, 299-307.

S. S. Gokhale and K. S. Trivedi (1999). A time/structure based software reliability model. *Annals of Software Engineering*, **8**, 85-121.

J.-Y. Park and T. Fujiwara (2006). Coverage growth functions for software reliability modeling. *Proceedings of 2006 Asian International Workshop on Advanced Reliability Modeling*, Pusan, Korea.

J.-Y. Park, T. Fujiwara and J.-H. Park (2005). A coverage function for arbitrary testing profile and its performance. *International Journal of Reliability and Applications*, **6**, 87-99.

J.-Y. Park, Y.-S. Hwang, and T. Fujiwara (2005). Integration of imperfect debugging in general testing-domain dependent NHPP SRGM. *International Journal of Reliability, Quality and Safety Engineering*, **12**, 493-505.

H. Pham (2003). Software reliability and cost models: perspectives, comparison, and practice. *European Journal of Operational Research*, **149**, 475-489.

H. Pham, and L. Nordmann (1997). A generalized NHPP software reliability model. *Proceedings of 3rd ISSAT International Conference on Reliability and Quality in Design*, **3**, 116-120.

H. Pham, and X. Zhang (1997). An NHPP software reliability model and its comparison. *International Journal of Reliability, Quality, and Safety Engineering*, **4**, 269-282.

H. Pham, and X. Zhang (2003). NHPP software reliability and cost models with testing coverage. *European Journal of Operational Research*, **145**, 443-454.

P. Piwowarski, M. Ohba and J. Caruso (1993). Coverage measure experience during function test. *Proceedings of 15th IEEE International Conference on Software Engineering*, Baltimore, MD, USA, 287-301.

A. T. Rivers and M. A. Vouk (1995). An empirical evaluation of testing efficiency during non-operational testing. *Proceedings of 4th Software Engineering Research Forum*, Boca Raton, Florida.

A. T. Rivers and M. A. Vouk (1998). Resource-constrained nonoperational testing of software. *Proceedings of 9th International Symposium on Software Reliability Engineering*, Paderborn, Germany.

M. A. Vouk (1992). Using reliability models during testing with nonoperational profile. *Proceedings of 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation*, 103-111.

M. A. Vouk and A. T. Rivers (2003). Construction of reliable software in resource constrained environment. Chapter 9 in *Case Studies in Reliability and Maintenance*, Editors: W. R. Blischke and D. N. Prabhakar Murthy, John Wiley and Sons, 205-231.

T. Yamamoto, S. Inoue, and S. Yamada (2004). A software reliability growth model with testing-coverage maturity process. *Proceedings of 10th ISSAT International Conference*, Las Vegas, NV, USA, 299-303.

X. Zhang, and H. Pham (2000). Comparison of nonhomogeneous Poisson process software reliability models and its application. *International Journal of Systems Science*, **31**, 1115-1123.

X. Zhang, X. Teng and H. Pham (2003). Considering Fault Removal Efficiency in Software Reliability Assessment. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, **33**, 114-120.