Statistical Certification of Software Systems

A. Di Bucchianico[1,2,4], J.F. Groote[1,2,3], K.M. van Hee[1,2] and R. Kruidhof[1]

[1] *Laboratory for Quality Software P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[2] *Department of Mathematics and Computer Science, Eindhoven University of Technology*

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[3] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

[4] *EURANDOM, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Email: `A.d.Bucchianico@tue.nl, J.F.Groote@tue.nl, K.M.v.Hee@tue.nl, R.Kruidhof@student.tue.nl`

Abstract

Common software release procedures based on statistical techniques try to optimise the trade-off between further testing costs and costs due to remaining errors. We propose new software release procedures where the aim is to certify that the software does not contain errors. The decision procedure is based on a mix of classical and Bayesian approaches to sequential testing. The procedure is applied to a new discrete-time model similar to the Jelinski-Moranda model. Because our procedure is generic, it may be extended to other models.

1 Introduction

Several statistical methods have been developed for assistance during software development processes (see Xie (2000) and Yamada (2001) for recent overviews). Many papers have been written on software reliability growth models that are helpful in predicting required additional testing effort before release. Software release itself has received less attention. Recent papers on this topic include Lee et al. (2001), Morali and Soyer (2003), Özekici and Soyer (2001) and Zacks (1995). Most papers interpret software release as an optimal stopping problem with a loss function that usually takes into account both costs of extra testing and customer costs due to undetected errors. A drawback to this approach is that such costs usually are difficult to assess or are negligible because of automated testing. Instead, we are interested in the following situation. Suppose software has been tested and we wish to certify that the software is reliable. Unlike in hardware reliability, it does not seem to be sensible to use metrics involving time in this case. Rather we wish to certify with a certain confidence, that the software does not contain errors. This problem seems not to have been studied in the software reliability, although a similar context appears in Becker and Camarinopoulos (1990) where a Bayesian method is proposed to sequentially estimate a failure rate.

This paper is organized as follows. In Section 2 we describe our model and define the common elements of our release procedures. A release procedure in case we exactly know θ, the detection probability of errors, is studied in Section 3. A generalization of this model where we assume in a Bayesian manner that θ has a prior distribution is studied in Section 4. We study a one-stage

testing procedure with unknown detection probability in Section 5. Section 6 contains an overview of on-going research.

2 The model and the release procedures

In this section we propose a simple model for software errors. This model will be used to illustrate our release procedures.

2.1 Details of the model

We assume that the number n of errors in a software system is unknown. The software system is subjected to a series of tests. A test is a walk through the state space of the system. Each walk has a finite length. A walk ends either if an end state is reached or if an error has occurred. We assume that the walks have been designed in such a way that each undetected error has a probability θ to be on the walk. So each error has probability θ to be detected during a test. This is not completely realistic, but may yield a reasonable approximation. When an error has been found, a test ends. Hence, in a test at most one error will be reported. When there are r remaining errors , the probability of not detecting any error in one test equals $(1-\theta)^r$. Knowledge about the software system to be tested (*e.g.*, maturity and level of concurrency) and the testing procedure (*e.g.*, quality of the tests) can be incorporated by choosing the parameter θ.

Let X_i $(1 \leq i \leq n)$ be the length of the ith test sequence, i.e. the number of tests to detect the ith error after the $(i-1)$st error has been repaired. Clearly X_i is geometrically distributed with success parameter $p_i = 1 - (1-\theta)^{n-i+1}$. For sake of brevity, we will often write $\varphi = 1 - \theta$. We drop the index i when we speak of an arbitrary test run length. Recall that if X is a geometrically distributed random variable with success parameter p, then the probability mass function is given by $P(X = t) = (1-p)^{t-1} p$ $(t \geq 1)$. Often it is more convenient to work with the survivor function $P(X \geq t) = P(X > t-1) = (1-p)^{t-1}$. Note that the software reliability literature almost exclusively deals with continuous-time models. The only exception that we are aware of is Yamada et al. (1986), which deals with a discrete time version of a nonhomogeneous Poisson process.

Recall that the failure rate of a nonnegative integer valued random variable X is defined as

$$\lambda(t) := P(X \leq t \mid X \geq t) = \frac{P(X = t)}{P(X > t-1)}.$$

Thus a geometrically distributed random variable X has failure rate $(1-p)^{t-1} p / (1-p)^{t-1} = p$. Hence, the geometric distribution has a constant failure rate, just like its continuous counterpart, the exponential distribution. In our model, we have a stochastic process with a nonhomogeneous failure rate λ:

$$\frac{P(X_i = t)}{P(X_i > t-1)} = \frac{(1-p_i)^{t-1} p_i}{(1-p_i)^{t-1}} = 1 - \varphi^{n-i+1} := \lambda_i$$

$$\lambda(s) = \lambda_i \text{ for } \sum_{\ell=1}^{i-1} X_\ell < s \leq \sum_{\ell=1}^{i} X_\ell.$$

This is similar to a discrete-time version of the Jelinski-Moranda model introduced in Jelinski and Moranda (1972); a true analogue would have geometric distributions with success parameters $(\lambda(n-i+1))^{-1}$.

2.2 General description of the release procedures

In practice unconstrained testing is often not feasible. We therefore assume that there are integers $k_1, k_2, \ldots, k_m$ such that instead of $X_1, X_2, \ldots, X_m$ we observe $T_1, T_2, \ldots, T_m$, where

$$T_i := \begin{cases} X_i & \text{if } X_i \leq k_i \\ k_i + 1 & \text{if } X_i > k_i \end{cases}.$$ This means that the test is terminated after testing at most k_i times.

When no errors are found in k_i tests, then $T_i := k_i + 1$. We use $k_i + 1$ to indicate whether a error has been found when during the k_ith test or whether no error was found at all. The probability mass function of a geometric random variable T with success parameter p that is truncated at k is given by

$$P(T = t) = \begin{cases} (1-p)^{t-1}\, p & \text{if } 1 \leq t \leq k \\ (1-p)^k & \text{if } t = k + 1 \\ 0 & \text{if } t > k + 1 \end{cases} \tag{1}$$

We now are ready to define the general form of our release procedures. We require a conclusion that all errors have been detected with confidence α. Typical values for α are 0.05 or 0.10. A straightforward idea would be to estimate n sequentially and test whether n is positive. However, it is known from the statistical literature that even for simple models the standard optimal estimators of n are unstable. Therefore we propose another method. After each observed failure detection, we have a new observed value t_i and test whether the number i of detected errors so far equals the total number of initial errors (n). Because of the one-sided nature of this testing, we test as follows. If $t_i = k_i$ and there is no observed error, then we conclude that $i = n$, i.e., there are no remaining errors left. If not, then we continue testing until we find either an error or we do not detect an error during k_{i+1} tests, in which case we declare that our system has no more errors. We now put this in a sequential hypothesis testing framework by using the following hypotheses:

$$H_{0,i}: \ n > i, \quad H_{1,i}: \ n = i.$$

The test statistic is T_i and the critical region is $\{t : t > k_i\}$. Note that whenever we do not reject $H_{0,i}$ for some i, we continue testing until we detect another error or we reach the critical bound. Since there are finitely many errors n and we assume perfect repair, our procedure always ends because we stop whenever we do not find an error in at most $n \max\{k_i : i = 1, 2, \ldots\}$ consecutive tests. Hence, we only have to guard ourselves against the possibility of falsely rejecting $H_{0,i}$ for some $i < n$, i.e., stop testing when there still are undetected errors. We control the probability of this situation (called a type I error in statistics) by choosing the k_i's. We do this in such a way that we do not assume anything on the number of initial errors. Hence, our procedure works for software systems of any size.

Alternatively, we could put our procedure in an optimal stopping framework with stopping time I and a special kind of loss function $\mathcal{L}$:

$$I = \min\{i : T_i > k_i\} \text{ and } \mathcal{L} = P_{\theta,n}(I \leq n). \tag{2}$$

Note that the event $\{I = n+1\}$ indicates that all n errors have been detected. Since we are interested in release of error-free systems, our loss function differs from commonly used loss functions (cf. the release procedures mentioned in the introduction).

3 Sequential procedure: known detection probability

In this section we assume that θ is known. This is not unrealistic, because we may have seen many similar systems the behaviour of which was observed beforehand. As remarked in Section 2, we must determine the values of k_i. As a first approximation we take a constant value k for the k_i's. As before, we write $\varphi = 1 - \theta$ and obtain

$$P_{\theta,n}(I \leq n) = 1 - P_{\theta,n}(I > n) = 1 - \prod_{i=1}^{n} \left(1 - \varphi^{(n-i+1)k}\right) = 1 - \prod_{j=1}^{n} \left(1 - \varphi^{jk}\right). \tag{3}$$

In order to have a type I error of at most α, we need to choose k such that $P_{\theta,n}(I \leq n) \leq \alpha$ for all n. Therefore we study the behaviour of $f_n = \prod_{j=1}^{n} \left(1 - \varphi^{jk}\right)$. Clearly f_n decreases in n, so k

must be chosen such that

$$\prod_{j=1}^{\infty} \left(1 - \varphi^{jk}\right) \geq 1 - \alpha. \tag{4}$$

There is no closed expression for the above infinite product. Note that the left-hand side of (4) is an increasing function of the discrete variable k. Hence, one could determine the minimal value of k such that inequality (4) holds by evaluating the left-hand side of (4) for an increasing sequence of values of k. This evaluation could be done numerically using software or by using sharp lower and upper bounds. For $0 < u < \frac{1}{2}$, the following sharp bounds may be derived from Taylor expansions for $\log(1 - u^j)$:

$$\exp\left(-\frac{u}{1-u^2}\left(1 + 2u\right)\right) \leq \prod_{j=1}^{\infty} \left(1 - u^j\right) \leq \exp\left(-\frac{u}{1-u^2}\left(1 + \frac{3}{2}u\right)\right) \tag{5}$$

There is another method for obtaining values of k such that (4) holds. This method only requires a small table of corrections to standard values of α. Moreover, this method yields a surprisingly accurate simple lower bound for k. Using the fact that we have geometric distributions, we have

$$P_{\theta,1}(T_1 > k) = \max_{n \in \mathbb{N}} P_{\theta,n}(T_1 > k) \leq \max_{n \in \mathbb{N}} P_{\theta,n}\left(\max_{1 \leq i \leq n} T_i > k\right) = 1 - \prod_{j=1}^{\infty} \left(1 - \varphi^{jk}\right).$$

Straightforward substitution shows that if $k_0 = \frac{\log \alpha}{\log \varphi} = \log_\varphi \alpha$, then $P_{\theta,1}(T_1 > k_0) = \alpha$. If we substitute $\log_\varphi \alpha$ for k into (4), then we obtain that the type I error of our procedure becomes $1 - \prod_{j=1}^{\infty} \left(1 - \alpha^j\right) \geq 1 - (1 - \alpha) = \alpha$. Since this value exceeds α, we see that k_0 is a lower bound. By numerically equating the upper bound in (5) to the desired type I error and then solving for α, we obtain a value for k that yields a procedure not exceeding the desired type I error. Table 1 gives values for typical type I errors and typical values of φ, while Table 2 shows that the lower bound is remarkably accurate.

α	$\widetilde{\alpha}$	$\varphi = 0.80$	$\varphi = 0.90$	$\varphi = 0.95$	$\varphi = 0.99$	$\varphi = 0.999$	$\varphi = 0.9999$
0.01	0.00985	21	44	91	460	4618	46201
0.05	0.04680	14	30	60	305	3061	30618
0.10	0.08877	11	23	48	241	2421	24216

Table 1: Values of $k = \log_\varphi \tilde{\alpha}$ for fixed detection probability $1 - \varphi$.

When testing is time consuming, it is important to study $S_I := \sum_{i=1}^{I} T_i$, the total number of

α	$\varphi = 0.80$	$\varphi = 0.85$	$\varphi = 0.90$	$\varphi = 0.95$	$\varphi = 0.99$	$\varphi = 0.999$	$\varphi = 0.9999$
0.01	21	29	44	90	459	4603	46050
0.05	14	19	29	59	299	2995	29956
0.10	11	15	22	45	230	2302	23025

Table 2: Values of lower bound $k_0 = {}^\varphi \log \alpha$ for fixed detection probability $1 - \varphi$.

tests in our release procedure. We wish to calculate $E(S_I)$ as metric for the performance of our release procedure. Using the Optional Stopping Theorem from probability theory and writing $p_i := 1 - (1 - \theta)^{n-i+1} = 1 - \varphi^{n-i+1}$, we obtain after some calculations that

$$E_{\theta,n}(S_I) = \sum_{\ell=1}^{n} \left(\frac{1 - \varphi^{(k+1)(n-\ell+1)}}{1 - \varphi^{n-\ell+1}} \prod_{i=1}^{\ell-1} \left(1 - \varphi^{(n-i+1)k}\right)\right) + (k+1) \prod_{j=1}^{n} \left(1 - \varphi^{jk}\right).$$

n	0	10	20	30	40	50	60	70	80	90	100
$E(S_I)$	3062	5795	6468	6871	7159	7385	7570	7728	7866	7988	8098

Table 3: Values of $E(S_I)$ for $\varphi = 0.999$ and $\alpha = 0.05$.

n	200	300	400	500	600	700	800	900	1000
$E(S_I)$	8839	9296	9636	9912	10149	10358	10548	10723	10886

Table 4: Values of $E(S_I)$ for $\varphi = 0.999$ and $\alpha = 0.05$.

α	0.01	0.05	0.10
$E(S_I)$	7496	5795	5020

Table 5: Values of $E(S_I)$ for $\varphi = 0.999$ and $n = 10$.

φ	0.80	0.85	0.90	0.95	0.99	0.999	0.9999
$E(S_I)$	34	43	63	120	583	5795	57911

Table 6: Values of $E(S_I)$ for $n = 10$ and $\alpha = 0.05$

The conclusion from Tables 3 and 4 is that the influence of the number of initial errors is relatively small, because a graph of these tables shows is rather flat. Table 5 shows that it does not make much difference to take a significance level of 5% instead of 10%, while Table 6 shows that expected total testing time is moderate for φ below 0.95 and increases exponentially for larger values of φ.

4   Sequential procedure: random detection probability

In this section we generalize the procedure of the previous section by replacing the fixed known detection probability by a random detection probability with known probability distribution (called prior distribution). We model this situation by assuming that θ is a continuous random variable on $[0, 1]$ with density q. This allows a more flexible modelling of knowledge about the software system and the testing system. As remarked in Section 2, we must determine the values of k_i. As in the previous section, we take a constant value k for the k_i's. A Bayesian one-stage testing procedure where k is updated using previous test results, is studied in Section 5. Using (3), we then obtain

$$P_n(I \leq n) = \int_0^1 P_{\theta,n}(I \leq n)\, q(\theta)\, d\theta = \int_0^1 \left(1 - \prod_{j=1}^n \left(1 - (1 - \theta)^{jk} \right) \right) q(\theta)\, d\theta.$$

In order to have a type I error of at most α, we need to choose k such that $P_n(I \leq n) \leq \alpha$ for all n. Like in the previous section, we use the sharp upper bound (5) for $\prod_{j=1}^\infty \left(1 - \varphi^{jk} \right)$:

$$\max_{n \in \mathbb{N}} P_n(I \leq n) \leq 1 - \int_0^1 \exp\left(-\frac{\varphi^k}{1 - \varphi^{2k}} \left(1 + 2\varphi^k \right) \right) q(1 - \varphi)\, d\varphi. \tag{6}$$

We saw in the previous section that a lower bound for k may be obtained by considering only the case that the last error needs to be detected. This lower bound turned out to be remarkably close to the optimal value of k. Therefore we try the same approach here as well. If there is only one

error left to be detected, then the distribution of tests to detect this error equals

$$P_1(T_1 > k) = \int_0^1 P_{\theta,1}(T_1 > k)\, q(\theta)\, d\theta = \int_0^1 (1-\theta)^k\, q(\theta)\, d\theta.$$

$\varphi = 1 - \theta$ uniformly distributed on	$[0.85, 1]$	$[0.95, 1]$	$[0.98, 1]$
k	169	507	1269
k_0	133	399	999

Table 7: Values of k and lower bound k_0 for φ uniformly distributed with $\alpha = 0.05$.

Table 7 shows that the lower bound is definitely less accurate than in the previous section, especially when the support of the parameter distribution is close to 1.

$\varphi = 1 - \theta$ uniformly distributed on	$[0.95, 0.99]$	$[0.95, 0.999]$	$[0.95, 1]$
k	141	374	507

Table 8: The influence of φ approaching 1 for $\alpha = 0.05$.

From Table 8 we conclude that the influence of letting the support of φ become too close to 1 is considerable.

Another suitable prior distribution of φ is the beta distribution. Table 9 shows some values for k where φ follows a beta distribution. The values for the parameters have been chosen such that the peak of the distribution is close to 1 (see Figures 1 and 2).
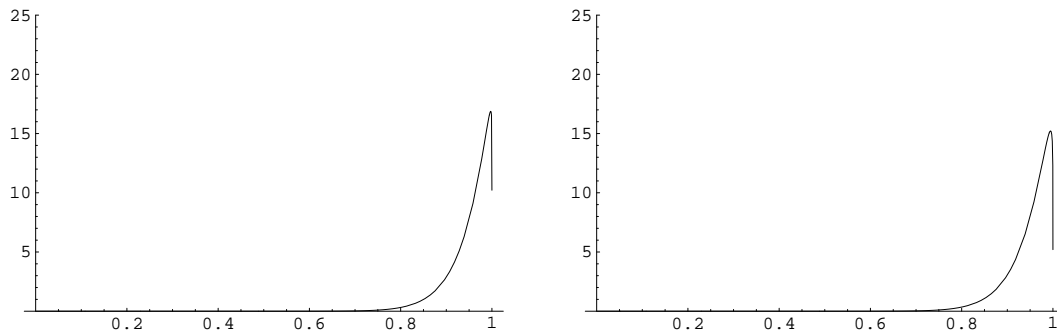

Figure 1: Beta distributions with parameters $20, 1.05$ and $20, 1.1$.

α, β	$20, 1.05$	$20, 1.1$	$30, 1.05$	$30, 1.1$
k	417	362	636	293

Table 9: Values of k for φ beta distributed with $\alpha = 0.05$.

5   Bayesian one-stage testing

We study in this section a Bayesian one-stage testing version of our release procedure. We have the following application in mind. Suppose that a company has performed testing on software and kept records. In order to obtain an official certificate that the software has no errors, the software must be tested by an official agency.
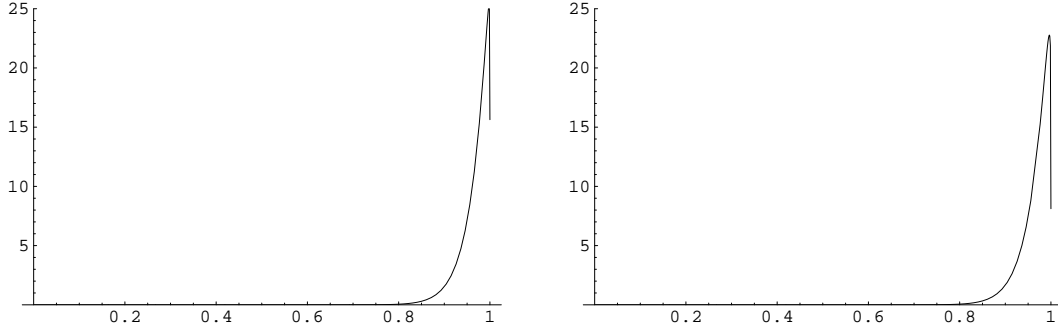
Figure 2: Beta distributions with parameters $30, 1.05$ and $30, 1.1$.

As said before we apply here a mixture of Bayesian and classical statistics. We do not assume a prior distribution for the total number of initial errors n as if often done, since we think it is unrealistic to assume that a good elicitation procedure for n exists in practice. Instead, we use a worst-case scenario: the maximal type I error as a function of n should be small. However, the detection probability of errors θ is assumed to be a random variable Θ with a prior distribution. In this way we are able to learn from former systems testing and also from the testing of the system under consideration. We assume that m errors haven been detected and that we know the number of tests needed to detect each of the m detected errors.

In order to have a type I error not exceeding α, we must find a critical value k such that

$$
\begin{aligned}
\alpha &\geq \max_{n \geq m+1} P_{\theta,n}(T > k \mid T_1 = t_1, \ldots, T_m = t_m) \\
&= \max_{n \geq m+1} \frac{\int_0^1 P_{\theta,n}(T > k, T_1 = t_1 \ldots, T_m = t_m) \, q(\theta) d\theta}{\int_0^1 P_{\theta,n}(T_1 = t_1 \ldots, T_m = t_m) \, q(\theta) d\theta}.
\end{aligned} \tag{7}
$$

Clearly the critical value k is a function of $t_1, \ldots, t_m$ and q only. Note that $T_1, T_2, \ldots, T_m$, and T are independent, given θ. Hence, the above inequality can be rewritten as

$$
\alpha \geq \max_{n \geq m+1} \frac{\int_0^1 \varphi^{(n-m)k+nw-v} \prod_{i=1}^m (1 - \varphi^{n-i+1}) \, r(\varphi) \, d\varphi}{\int_0^1 \varphi^{nw-v} \prod_{i=1}^m (1 - \varphi^{n-i+1}) \, r(\varphi) \, d\varphi}, \tag{8}
$$

where $r(\varphi) = q(1 - \varphi) = q(\theta)$ and

$$
w_i = t_i - 1, \quad w = \sum_{i=1}^m w_i, \quad v = \sum_{i=1}^m (i - 1) w_i. \tag{9}
$$

In order to obtain computable lower bounds for k, we apply a general form of Chebyshev's inequality (for a detailed discussion, we refer to Chapter 9 of Mitrinović, Pečarić, and Fink (1993)). We then obtain

$$
\max_{n \geq m+1} \frac{\int_0^1 \varphi^{(n-m)k+nw-v} \prod_{i=1}^m (1 - \varphi^{n-i+1}) \, r(\varphi) \, d\varphi}{\int_0^1 \varphi^{nw-v} \prod_{i=1}^m (1 - \varphi^{n-i+1}) \, r(\varphi) \, d\varphi} \leq \max_{n \geq m+1} \frac{EY^{n(w+k)-(mk+v)}}{EY^{nw-v}}, \tag{10}
$$

where Y denotes a random variable with density r. The right-hand side of (10) can be shown to be increasing in n for fixed k in case of a uniform distribution on a subset of $[0, 1]$ or a beta distribution. This fact allows an explicit condition that k must satisfy as we will now show.

Computation of k in case of a uniform distribution The jth moment of the uniform distribution on $[\ell, 1]$ $(0 \leq \ell < 1)$ equals $\frac{1}{(1-\ell)(j+1)} \left(1 - \ell^{j+1}\right)$. A long calculation shows that (10) is maximal for $n = m + 1$. Hence, we need to choose k such that

$$
\frac{(m+1)w - v + 1}{k + (m+1)w - v + 1} \frac{1 - \ell^{k+(m+1)w-v+1}}{1 - \ell^{(m+1)w-v+1}} = \frac{EY^{k+(m+1)w-v}}{EY^{(m+1)w-v}} \leq \alpha.
$$

7

Computation of k in case of a beta distribution The jth moment of the beta distribution with parameters (γ,δ) equals $\frac{\gamma^{[j]}}{(\gamma+\delta)^{[j]}}$ where $\gamma^{[j]} = \gamma(\gamma+1)\dots(\gamma+j-1)$ denotes the Pochhammer symbol. It is easy to show that (10) is maximal for $n = m+1$. Hence, we need to choose k such that

$$\frac{\gamma^{[k+(m+1)w-v]}}{\gamma^{[(m+1)w-v]}} \frac{(\gamma+\delta)^{[(m+1)w-v]}}{(\gamma+\delta)^{[k+(m+1)w-v]}} = \frac{EY^{k+(m+1)w-v}}{EY^{(m+1)w-v}} \leq \alpha.$$

6 Conclusions and future work

The present paper is a first attempt to build a decision procedure for software certification of error-free software. The procedure is simple to set up and only need values of the critical parameter k. The values for k are reasonable if we may assume that the detection probability stays away from 0. The procedure does not assume anything on the initial number of errors. Hence, the actual statistical type I error of our procedure (the probability of incorrectly certifying that the software is error-free) when used in practice may be much lower than the guaranteed type I error.

We are currently extending the procedure in several directions, including situations with different detection probabilities and grouped data (*i.e.*, detection times are only available per day or week). Preliminary simulations show that the Bayesian one-stage testing performs reasonably well when we repeatedly update the value of k. Finally, we are exploring ways to incorporate software structure like modules in our models.

References

G. Becker and L. Camarinopoulos. A Bayesian estimation method for the failure rate of a possibly correct program. *IEEE Trans. Soft. Eng.*, 16(11):1307–1310, 1990.

Z. Jelinski and P. Moranda. Software reliability research. In W. Freiberger, editor, *Statistical Computer Performance Evaluation*, pages 465–497. Academic Press, 1972.

C.H. Lee, K.H. Nam, and D. H. Park. Optimal software release policy based on Markovian perfect debugging model. *Comm. Statist. Theory Methods*, 30(11):2329–2342, 2001. International Conference on Statistics in the 21st Century (Orono, ME, 2000).

D.S. Mitrinović, J.E. Pečarić, and A.M. Fink. *Classical and new inequalities in analysis*, volume 61 of *Mathematics and its Applications (East European Series)*. Kluwer Academic Publishers Group, Dordrecht, 1993.

N. Morali and R. Soyer. Optimal stopping in software testing. *Naval Res. Logist.*, 50(1):88–104, 2003.

S. Özekici and R. Soyer. Bayesian testing strategies for software with an operational profile. *Naval Res. Logist.*, 48(8):747–763, 2001.

M. Xie. Software reliability models—past, present and future. In *Recent advances in reliability theory (Bordeaux, 2000)*, Stat. Ind. Technol., pages 325–340. Birkhäuser Boston, Boston, MA, 2000.

S. Yamada. Software reliability models. In S. Osaki, editor, *Stochastic Models in Reliability and Maintenance*, pages 253–280. Springer, 2001.

S. Yamada, S. Osaki, and H. Narihisa. Discrete models for software reliability evaluation. In *Reliability and quality control (Columbia, Mo., 1984)*, pages 401–412. North-Holland, Amsterdam, 1986.

S. Zacks. Sequential procedures in software reliability testing. In *Recent advances in life-testing and reliability*, pages 107–126. CRC, Boca Raton, FL, 1995.