TR-91-013

ON OPTIMAL STOPPING RULES IN SOFTWARE RELIABILITY

ON OPTIMAL STOPPING RULES IN SOFTWARE RELIABILITY

Mark C. K. Yang
Institute of Information Science
Academia Sinica, Nankang, Taiwan, ROC, and
Department of Statistics
University of Florida
Gainesville, Florida 32611

Anne Chao
Institute of Statistics
National Tsing Hua University
Hsin-Chu, Taiwan, R.O.C. 30043

On Optimal Stopping Rules in Software Reliability

By Mark C. K. Yang and Anne Chao

Abstract — Stopping rules for software testing are determined by the cost of testing and the penalty of the unremoved bugs that may be encountered by future users. Unlike most previous criteria, the damage is not assessed by the remaining bugs, but by the chance that they may be encountered in the future when the software is run. No assumptions on the bug distribution is assumed. The stopping rules are adapted from the theoretical optimal stopping rule which requires a known distribution of the bugs. Our simulation study shows that under a wide variety of situations, the adaptive rules are nearly optimal.

Index Terms -Debugging, optimal stopping rule, random testing, software reliability, software testing.

I. INTRODUCTION

Software testing, or debugging, is one of the most important components in software development. It has been estimated that in many projects, the time accounted for debugging can be around 50% of the total development effort[12]. There is an obvious question in the debugging process, that is when to stop. One naive answer is, of course, the process continues until there are no bugs (errors) in the program. Though formal verification language[15] is designed for this purpose, its implementation can be very time consuming. Thus, except for extremely sensitive programs, such as those related to the national security, formal verification is too expensive to apply. For most commercial software, the release requirement is usually not 100% error free, but an acceptable error rate. Naturally, there is always some reluctance for a producer to define and announce an acceptable error rate for his product. Even if the acceptable rate is set, to determine when a debugging process has reached this stage is again difficult. The best bet is often an *estimate* of the future error rate. However, the accuracy of the estimate may not be very high, depending on the estimation formula, and more seriously, on the assumptions that the formula is based upon. Thus, to examine the applicability of a software testing and reliability evaluation procedure, one needs to check 1) the assumption made on the error rate configuration, 2) the sampling and testing procedure, and 3) the decision criterion. Listed below are some of the commonly used assumptions, sampling procedures, and decision criteria.

A) Assumptions:

A1: There are m (unknown) bugs in the program with occurrence (failure) rates $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m > 0$.

A2: There is a known distribution of λ. For example, equal failure rate (EFR) $\lambda_1 = \lambda_2 = \dots = \lambda_m$.

Remark 1. The meaning of failure rate λ_i is that suppose n programs are run (tested), then on the average $n\lambda_i$ of them will encounter bug i. Now if n is proportional to the testing time t, then the number of times bug i is encountered before time t is approximately a Poisson distribution with mean $\lambda_i \rho t$, where ρ is the proportional constant $n=\rho t$. Or, more realistically, the time should be measured by the testing effort $t=n/\rho$, instead of the calendar time. Throughout this paper, n and t are treated as aliases.

Remark 2. The restriction $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m > 0$ does not lose any generality.

B) Sampling Procedures:

T1: Only one random sample is used for testing. The sample size has to be determined by some prior knowledge of λ [3].

T2: Sequential test. Random samples are taken sequentially and the testing procedure stops when a certain criterion is met.

C) Decision Criteria:

D1: With $(1 - \alpha)$ confidence that the remaining number of bugs is less than or equal to m_r.

D2: With $(1 - \alpha)$ confidence that the remaining failure rate is less than or equal to Λ_r.

D3: Assign c_1 as the cost of testing one program and c_2 as the cost of one remaining bug. Find a procedure that minimize the total cost.

D4: Assign c_1 again as the cost of testing one program. Let c_2 be the cost of ruining one customer's output due to the bugs and M be the number of programs that are expected to run in the future. Find a procedure that minimize the total cost.

Remarks: 1) Under A2 EFR, D1 and D2 are equivalent and so are D3 and D4.

2) All the constants; α, m_r, c_1, c_2, M, and Λ_r have to be determined by the tester.

Table I gives some of the existing stopping rules. Since our interest is in sequential testing, only those methods that are emphasized in T2 are included.

(Insert Table I)

To evaluate the procedures in Table I, we first consider the assumptions. Unless we have some strong feeling on the distributions on λ, A1 should be the most general and reasonable choice. Readers who are interested in various modeling on λ can refer to [1, 7, 11, 9]. As for the sampling procedure, it is apparent that T2 is more flexible and requires no prior knowledge on λ. It is also quite intuitive that D1 and D3 are less reasonable than D2 and D4. Depending on the occurrence rates, a small number of bugs can cause more damage than a large number of bugs. Now, among D2 and D4, we feel that D4 is more practical because it is harder for a producer to determine the confidence bound $1 - \alpha$ and Λ_r than for him to have some quantitative measure on the two costs. Actually, it is easy to argue that if the cost c_2 is much larger than c_1 then one should do more testing, and vice versa. But this fact cannot be easily converted to α in the D2 framework. Actually, the two constants c_2 and M in D4 are inseparable and can be considered as one constant. For no matter what the value of c_2 is, the real cost of damage depends on how much this program is used. Here M can also be interpreted as the expected number of programs to be run in the lifetime or the time before the next revision of the software. In Table I, only Brown, Maghsoodloo, and Deason[3] consider D4. But unfortunately, not only are the constants such as E and N in (T.3) difficult to estimate, but also, no debugging procedure is reflected in the derivation. The number of bugs are the same before and after the testing. Thus, we feel there are still gaps before one can actually use the stopping rules in [3].

In this paper, we try to find stopping rules with the assumption A1, sampling procedure T2, and decision rule D4 under two possible debugging schemes. In the next section, theoretical optimal stopping rules are derived for the case when the λ's are known and they are adapted to the situation when the λ's are unknown. Those properties of the adaptive rules that cannot be obtained through mathematical analysis are simulated in Section III. Some concluding remarks are given in the final section IV.

II. OPTIMAL AND ADAPTIVE STOPPING RULES

In this section, only the most reasonable cases are presented. Possible generalizations to more complicated situations are given in §IV. The assumption on λ is

A1 and the sample procedure is T2 with a sample size of N in each testing period, $N \geq 1$. Here we do not restrict to the case $N = 1$ as in the usual sequential analysis, because for large programs, the testing and debugging may not be done simultaneously. Debugging is performed only when a large number of programs has been tested. Two options are allowed for debugging. They are: 1) standard debugging where bugs are removed after each testing period, and 2) recapture debugging[13] where the discovered bugs are bypassed but a record is still kept every time they are encountered.

Let I(x) be the indicator function

$$I(x) = \begin{cases} 1 & \text{if x is true} \\ 0 & \text{if x is false,} \end{cases}$$

$X_i(n) =$ number of times that the ith bug is encountered at the end of the nth test period,

and the remaining failure rate

$$U(n)= \sum_{i=1}^{m} \lambda_i I(X_i(n)=0).$$

Then if the program is released after the nth testing period, the cost is

$$C(n) = c_2 M \cdot U(n) + c_1 nN. \tag{2.1}$$

Our purpose is to find the optimal stopping rule ψ such that

$$EC(\psi) \leq EC(\tau),$$

for any stopping rule τ. Here E denotes the expectation. Note that a stopping rule is a decision that depends only on the sampling information from the past, not the future. To put it in the usual notation, a stopping rule τ is a random variable such that the event (τ=n) $\epsilon\ \mathcal{F}_n$, where $\mathcal{F}_n$ is the sigma field generated by all the previous samples up to n. To find the optimal stopping rule, we first assume that all the λ's are known and use Theorem 3.3 in Chow, Robbins, and Siegmund[5]. In order to follow the theorem more easily, we let the payoff function g(n)=–C(n) and the equivalent optimal stopping rule now is to find ψ such that

$$Eg(\psi) \geq Eg(\tau), \tag{2.2}$$

for any stopping rule τ. Without loss of generality, we may let c_1=1 and c_2M=c. Hence,

$$g(n) = - cU(n) - nN. \tag{2.3}$$

Using our notation, we restate Theorem 3.3 of [5]. If the set

$$A_n = \{ \ E(g(n+1) \mid \mathcal{F}_n) \leq g(n) \} \tag{2.4}$$

is monotonically increasing with respect to n and

$$\liminf_n \int_{\psi>n} g^+(n)dP = 0, \tag{2.5}$$

then (2.2) is true for all τ satisfying

$$\liminf_n \int_{\tau>n} g^-(n)dP = 0, \tag{2.6}$$

where ψ is

$$\text{the first } n \geq 1 \text{ such that } g(n) \geq E(g(n+1) \mid \mathcal{F}_n). \tag{2.7}$$

Because $g^{+}(n)=0$ for all n, (2.5) holds in a trivial way. To show (2.4), note that

$$E(g(n+1) \mid \mathcal{F}_n) \leq g(n)$$

$$\Leftrightarrow -c \sum_{\{i:\, X_i(n) = 0\}} \lambda_i\, P\{ I(X_i(n+1) = 0) \mid I(X_i(n) = 0)\} - (n+1)N \leq -cU(n) - nN$$

$$\Leftrightarrow c \sum_{i=1}^{m} \lambda_i[1-(1-\lambda_i)^N]\, I\,(X_i(n) = 0) \leq N$$

$$\Rightarrow c \sum_{i=1}^{m} \lambda_i[1-(1-\lambda_i)^N]\, I\,(X_i(n+1) = 0) \leq N$$

$$\Rightarrow A_n \subset A_{n+1}\,.$$

Thus, (2.4) is proven. Since $U(n) \leq 1$, $g^{-}(n) = cU(n) + nN \leq c + nN$, and

$$\int_{\tau>n} g^{-}(n)dP \leq (c+nN)P\{\tau>n\} \rightarrow 0,$$

as $n \rightarrow \infty$, if $E\tau < \infty$. Thus, we have shown that ψ is the optimal stopping rule for all stopping rules with finite expectation. Suppose $E\tau = \infty$, then $Eg(\tau) \leq -E\tau = -\infty$. Because the stopping rule that stops with $\psi \equiv 1$ has expectation greater than $-(c+N)$, this τ cannot be optimal. We have proven (2.2) for all τ. To rewrite (2.7) explicitly, ψ is to stop at

$$\text{the first } n \geq 1 \text{ such that } \sum_{i=1}^{m} \lambda_i[1-(1-\lambda_i)^N]\, I(X_i(n)=0) \leq N/c. \qquad (2.8)$$

For small λ's, a good approximation for (2.8) is

$$\text{the first } n \geq 1 \text{ such that } \sum_{i=1}^{m} \lambda^2{}_i I(X_i(n)=0) \leq 1/c. \qquad (2.8')$$

Since the λ's are unknown, the ψ defined in (2.8) cannot be put into practice, but it tells us that if there is a good estimate of the left side of the inequality in (2.8) or (2.8′), we may be close to the optimal stopping rule. Moreover, if any stopping rule that can almost reach the optimal value EC(ψ) obtained from ψ when the λ's are known, it must be nearly optimal. From the simulation study to be presented in the next section, many nearly optimal situations are identified.

Let $\theta = E \sum_{i=1}^{m} \lambda^2_i I(X_i(n)=0)$. Then it can be shown that

$$\theta = \sum_{i=1}^{m} \lambda^2_i (1-\lambda_i)^{nN} . \qquad (2.9)$$

It is known in the literature (eg. [2], [4], [17]) that

$$E \sum_{i=1}^{m} I(X_i(n)=2) = \frac{nN(nN-1)}{2} \sum_{i=1}^{m} \lambda^2_i (1-\lambda_i)^{nN-2}.$$

By ignoring the small difference between $(1-\lambda_i)^{nN-2}$ and $(1-\lambda_i)^{nN}$, we can estimate θ by

$$\frac{2}{nN(nN-1)} \sum_{i=1}^{m} I(X_i(n)=2) \equiv \frac{2}{nN(nN-1)} B_n,$$

where B_n is the number of doubletons, i.e., the number of bugs that have been encountered exactly twice up to stage n. Thus, a reasonable adaptive rule $\hat{\psi}$ for the recapture debugging procedure is to stop at

$$\text{the first n such that } \frac{2}{nN(nN-1)} B_n \leq 1/c. \qquad (2.10)$$

The expected cost of this rule is denoted by EC($\hat{\psi}$).

When the standard debugging procedure is used, the number of doubletons up to stage n has to be estimated, because all the previous bugs before stage n have been removed. Let s_n and b_n denote the number of singletons and doubletons discovered at test period n. They are observable. From

$B_n =$ (Those bugs in B_{n-1} that are not encountered in period n) +

(Those bugs in S_{n-1} that are encountered exactly once in period n) + b_n,

where S_n denotes the number of singletons encountered up to stage n, S_n and B_n can be estimate recursively by

$$\hat{B}_n = \hat{B}_{n-1}\left(1-\frac{2}{N(n-1)}\right)^N + \hat{S}_{n-1}\left(1-\frac{1}{N(n-1)}\right)^{N-1}/(n-1) + b_n \qquad (2.11a)$$

$$\hat{S}_n = \hat{S}_{n-1}\left(1-\frac{1}{N(n-1)}\right)^N + s_n. \qquad (2.11b)$$

The two formulae are derived from the maximum likelihood principle. For example, the first term of (2.11a) is obtained by replacing the λ_i in

$$\sum_{i=1}^{m}(1-\lambda_i)^N I(X_i(n)=2)$$

by $2/[N(n-1)]$, for the ith error appears exactly twice in a sample of size N(n-1). In summary, for the standard debugging procedure, the stopping rule $\tilde{\psi}$ is to stop at

$$\text{the first n such that } \frac{2}{nN(nN-1)}\hat{B}_n \le 1/c. \qquad (2.12)$$

Again, we denote the cost under this rule by $EC(\tilde{\psi})$.

Analytic study on the performance of (2.10) and (2.12) seems to be very difficult. Simulations are used to evaluate their performances.

III. SIMULATION STUDY

The stopping rules; optimal (2.8), approximation (2.8′), adapted to recapture debugging procedure (2.10), and adapted to the standard debugging procedure (2.12) are compared. In standard software development, the failure rate should not be very high at the testing stage. Three values, 0.10, 0.05, and 0.01 are assigned for the total failure rate $T_\lambda \equiv \sum_{i=1}^{m} \lambda_i$, with m=100. Four configurations for λ are used;

a) rapidly decreasing λ (exponential rate): $\lambda_i = K/2^i$, i=1,2,...,m;

b) moderately decreasing λ (Zipf's Law): $\lambda_i = K/i$, i=1,2,...,m;

c) slowly decreasing λ (constant): $\lambda_i = K$, for all i=1, . . .,m;

d) random λ (following [16]): $\lambda_i = K\ U_i$, U_i is a random number, i=1,...,m,

where K is the normalization constant so that $\sum_{i=1}^{m} \lambda_i = T_\lambda$.

Since T_λ is small, small N will make the test very ineffective. We choose N=100, 300, 500. Since N=100 is somètimes too small for the case $T_\lambda=0.01$, we hold our decision if there is no doubletons before the 5th period. Similarly, we hold our decision for N=300 and 500 if there no duobletons in the first period. The value $c=c_2M/c_1$ can vary considerably due to different real situation. We feel that $c_1=1$, $c_2=100$, and $M=10^4$ is a reasonable middle ground. Thus, three values, $c=10^5$, 10^6, and $10^7$are used. One hundred simulations were done for each combination of c, N, and T_λ.

(Insert TableII(a-d))

The simulation results for $c=10^6$ are given in TableII (a-d). The other two c's give similar results. The differences between (2.8) and (2.8′) are so small in every case that the figures for (2.8′) are not worth reporting. In the Tables, the average cost and the average stopping time (in parentheses) associated with each combination of T_λ and λ distributions are presented. The following facts have been observed.

1) The first thing that surprises us is that there is little difference between (2.10) and (2.12). After detailed check into the stopping process, we found that this was due to large variation in B_n, the number of doubletons. Singletons are more stable in the sample. Thus, the estimated $\hat{B}_n$ from singletons can be as effective as the doubletons. This is also a reflection of the effectiveness of the recursive formula (2.11). We have tried some other ad hoc rules based on the singletons such as to stop at the first n such that $S_{n-1}/N(n-1) - S_n/Nn < 1/c$. Their performances are not comparable with (2.12).

2) It is actually unfair to compare (2.10) and (2.12) with (2.8), because in (2.8) all the λ's have to be known. There is a tremendous prior information difference between (2.8) and the two adaptive stopping rules. However, the simulations show that in most situations, especially Table II (b-d), the adaptive methods perform extremely well. It is unlikely that in these situations any other stopping rule can beat them without any prior information on λ. At least we can say that they are nearly optimal.

3) From the expected cost point of view, the initial total failure rate T_λ has less influence than the sizes of λ. Naturally, it is easy to debug one big $\lambda=T_\lambda=0.1$ than to debug 100 small equal λ's with a total $T_\lambda=0.01$. However, it has also been confirmed by our other simulations that if $T_\lambda < T'_\lambda$ and the λ's in the T_λ set is a subset of those in the T'_λ set, then the expected cost in the first situation is smaller than that in the second. Take Table II(c) for example, if the bugs in $T_\lambda=0.05$ were $\lambda_1=\lambda_2=\cdots=\lambda_{50}=0.001$, a subset of $T_\lambda=0.10$ with 100 bugs, then the cost for debugging $T_\lambda=0.05$ is smaller.

4) In Table II(a), we see a bigger discrepancy in costs between (2.10), (2.12), and

(2.8). Also in the case not shown when $c=10^5$ and $T_\lambda=0.01$, there are high increases in the cost for the slowly decreasing λ. Actually it is easy to identify these situations. For example, when $c=10^5$ and $T_\lambda=0.01$, we have $\Sigma\ \lambda_i^2 = 10^{-6} < 1/c$. Obviously, if this fact is known, no testing is the best strategy. But under the situation that the λ's are unknown, it will take a considerable number of testing samples to discover this fact. Thus, the adaptive methods cost more than the optimal rule (2.8). Another situation, such as the rapidly decreasing λ case, is that although $\Sigma\ \lambda_i^2 < 1/c$ is not true for all the λ's in the beginning, the λ's are dominated by a few large ones and once they are removed, $\Sigma\ \lambda_i^2 < 1/c$ is satisfied by the rest of the λ's. Since the large λ's can be discovered pretty easily, they can be removed in the very beginning. The debugging process can then be stopped because the λ's are known. The adaptive methods again have to identify this fact at considerable cost.

5) From the four tables, it is evident that the final costs vary little due to the test size N. Also, the stopping times are inversely proportional to the test size N, i.e., the total number of tested cases before stop is basically invarinat under different test sizes.

SUMMARY AND CONCLUDING REMARKS

Adaptive optimal stopping rules for software testing are derived based on the cost of testing and the penalty of the future damage that the bugs may induce. The rules are given in (2.10) and (2.12) depending on whether the recapture or standard debugging procedures are used. A few additional observations from the study are listed below.

1) As stated in Remark 1 in C, §1, D3 and D4 are equivalent when the λ's are equal. In this case, (2.8) is equivalent to (2.1) in Rasmussen and Starr[14]. We also repeated the simulation for the cases they considered. Our results confirm their results. However, their $T_\lambda=1.0$, designed for biological research, is unrealistic for software testing.

2) In the present study, the testing size N is assumed to be the same in all the testing stages. An interesting question would be what happens if we vary N. One thing we noticed is that the proof of the optimality of (2.8) is no longer valid. It seems to be a significant contribution if the tester can choose the optimal sample size at each stage.

3) From the derivation of (2.8), we can extend the result to a more general cost function i.e., let the cost for doing x tests be f(x). Then if $\Delta f(x) \equiv f(x+1) - f(x)$ is a nondecreasing function, then Theorem 3.3 of [5] holds and the optimal stopping rule becomes to stop

$$\text{at the first } n \geq 1 \text{ such that } \sum_{i=1}^{m} \lambda_i[1-(1-\lambda_i)^N]\, I(X_i(n)=0) \leq [f((n+1)N) - f(nN)]/c.$$

The assumption of $\Delta f(x)$ being nondecreasing is reasonable when delay in releasing the software is considered as a cost.

APPENDIX (FORMULAE FOR EXISTING STOPPING RULES LISTED IN TABLE 1)

In order to be more compatible with the formulae in the original papers, some of the notations may not be consistent with the notation defined in the text. But we try to keep them as clear as possible.

(T.1) Let T_i = time that the ith bug is discovered, m_t = total number of bugs discovered at time t, and Λ_ℓ= max failure rate acceptable. Stop testing at the first t when

$$\sum_{i=1}^{m_t} \frac{e^{-t/T_i}}{T_i(1-e^{-t/T_i})} + \gamma_\alpha \left[\sum_{i=1}^{m_t} \frac{e^{-t/T_i}}{T_i^2(1-e^{-t/T_i})^2}\right]^{\frac{1}{2}} \leq \Lambda_\ell,$$

where γ_α is a probability tail constant adapted for sequential testing at significance level α. The author suggests to use $\gamma_\alpha = 3$ for $\alpha = 0.05$.

(T.2). [14]: Stop at the first $n \geq 10$ such that

$$c_2 \text{ (\# of singletons)/(\# of cases tested) } \leq c_1. \qquad \text{(T.2)}$$

Sample size at each testing period is N=1. [2] extends the results to general N, but without giving any useful stopping rule. They also obtained another variation of (T.2).

(T.3). N = # of program population size, c_1, c_2, M = Same definition as those in the text, E = # of initial errors, $p = E/N$.

One time test: Test t_0 programs, where

$$t_0 = [\ell n c_1 - \ell n c_2 p + \ell n \ell n(1-p)]/\ell n(1-p).$$

Sequential test: Solve t in the equation:

$$c_1(N+t)^2 + c_2E(1-p)^t M[(N+t)\ell n(1-p) - 1] = 0.$$

(T.4). Let T_i, m_t be defined by (T.1), $t_i = T_i - T_{i-1}$, $T = \sum_{i=1}^{m_t} t_i$,

$k = \sum_{i=1}^{m_t} (i-1)t_i$, $\hat{\phi} = m_t/(Tm - k)$. Let $\hat{m}$ be the solution that maximizes

$$\mathcal{L}(m) = \hat{\phi} \prod_{i=1}^{m_t} (m-i+1)e^{-(m-i+1)\hat{\phi}t_i}.$$

If $\hat{m} >> m_t$, continue sampling, or else compare (graphically as a function of m),

$$R_1(m) = \prod_{i=1}^{m_t} \left(\frac{m-i+1}{\hat{m}-i+1}\right) \exp(-(m-\hat{m})\hat{\phi}t_i) \text{ and}$$

$$R_2(m) = \exp(-\tfrac{1}{2}(\hat{m} - m)^2/\sigma^2), \text{ where}$$

$$\sigma^2 = \frac{m_t}{m_t \sum_{i=1}^{m_t} \left(\frac{1}{m-i+1}\right)^2 - \left(\sum_{i=1}^{m_t} \frac{1}{m-i+1}\right)^2}.$$

If $R_1(m)$ and $R_2(m)$ are close, stop, or else take more samples.

Remark: $\hat{m}$ is the m.l.e. of m. Intuitively, if $m_t = \hat{m}$, we got all the bugs. Stop.

(T.5). Start with j=1. When the j distinct errors are detected, compute

$$k_j = [\log \alpha / \log (j/(1+j))] + 1.$$

Continue sampling until k_j old bugs are found before any new bugs appear. (D1 decision rule with $m_r = 0$)

(T.6). Extension of [13] to the case when several types of bugs are allowed. The decision rule is based on D1 with given α and $m_r = 0$. However, only rules for no more than two types of bugs are studied in detail.

Affiliations of authors: M. C. K. Yang is with with the Department of Statistics, University of Florida, Gainesville, Florida 32611.

A. Chao is with the Institute of Statistics, National Tsin-Hua University, Hsin-Chu Taiwan, R.O.C. 30043

REFERENCES

[1] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability prediction," *IEEE Trans. Software Eng.*, vol. SE-12, no. 9, pp. 950-966, Sept. 1986.

[2] P. K. Banerjee and B. K. Sinha, "Optimal and adaptive strategies in discovering new species," *Sequential Analysis*, Vol. 4, pp. 111-122, 1985.

[3] D. B. Brown, S. Maghsoodloo, and W. H. Deason, "A cost model for determining the optimal number of software testing cases," *IEEE Trans. Software Eng.*, Vol. Se-15, no. 2, pp. 218-221, Feb. 1989.

[4] A. Chao, "On estimating the discovering of a new species," *Annals of Statistics*, Vol. 9, pp. 1339-1342, 1981, Correction, 10, p. 1331, 1982.

[5] Y. S. Chow, H. Robbins, and D. Siegmund, *Great Expectation: The theory of optimal stopping*, Houghton Miffon Co., 1971.

[6] E. H. Forman and N. D. Singpurwalla, "An empirical stopping rule for debugging and testing computer software," *J. Am. Stat. Assoc.*, Vol. 72, pp. 750-757, Dec. 1977.

[7] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance resources," *IEEE Trans. Rel.*, Vol. R-

33, pp. 176-183, 1984.

[8] I. B. J. Goudie, "A likelihood-based stopping rule for recapture debugging," *Biometrika*, Vol. 77, 1, pp. 203-206, 1990.

[9] N. Langberg and N. D. Singpurwalla, "A unification of some software reliability models," *SIAM J. Sci. Stat. Comput.*, Vol. 6, no. 3, pp. 781-790, July 1985.

[10] B. Littlewood, "Stochastic reliability growth: a model for fault-removal in computer programs and hardware designs," *IEEE Trans. Rel.*, Vol. R-30, pp. 313-320, Oct. 1981.

[11] B. Littlewood and J.L. Verrall, "A Bayesian reliability growth model for computer software," *J. Roy. Statist. Soc. C*, Vol. 22, pp. 332-346, 1973.

[12] G. J. Myers, *Composite/Structure Design*. New York: Van Nostrand Reinhold, 1978.

[13] T. K. Nayak, "Estimating population size by recapture sampling," *Biometrika*, Vol. 75, no. 1, pp. 113-20, 1988.

[14] S. L. Rasmussen and N. Starr, "Optimal and adaptive stopping in the searching for new species," *J. Am. Stat. Assoc.*, Vol. 74, pp. 661-667, Sept. 1979.

[15] D. J. Richardson and L. A. Clarke, "Partition analysis: A method combining testing and verification," *IEEE Trans. Software Eng.*, Vol. SE-11, No. 12, pp. 1477-1489, Dec. 1985.

[16] S. M. Ross, "Software reliability: The stopping rule problem," *IEEE Trans. Software Eng.*, Vol. SE-11, No. 12, pp. 1472-1476, Dec. 1985.

[17] N. Starr, "Linear estimation of the probability of discovering a new species," *Annals of Statistics*, Vol. 7, no. 3, pp. 644-652, 1979.

Acknowledgment: This work was done when the first author was with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, under a grant from the National Science Council of the Republic of China.

TABLE I

SOME EXISTING STOPPING RULES AND RELEASE ERROR ESTIMATION IN SOFTWARE TESTING

Reference #	Assumptions	Testing Procedure	Decision Criterion	Estimation (stopping) Procedure
[16]	A1	T2	D2	(T.1) *
[14], [2]	A1	T2	D3	(T.2)*
[3]	A2 (EFR)	T1 and T2	D3 or D4	(T.3a), (T.3b)*
[6]	A2 (EFR)	T2	D1	(T.4)*
[13]	A2 (EFR)	T2	D1	(T.5)*
[8]	A2 (general)	T2	D1	(T.6)*

* All the formulae are given in APPENDIX

TABLE II (a) (FOR RAPIDLY DECREASING λ)

SIMULATION RESULTS ON THE COST INCREASES BY THE TWO ADAPTIVE METHOD (UNIT OF COST = 10^3) AND THE AVERAGE STOP TIME (IN PARENTHESES)

		T_λ		
		0.1	0.05	0.01
	OPTIMAL	1.93 (8.45)	1.93 (8.45)	2.00 (11.5)
N=100	(2.10)	3.53 (7.93)	3.53 (7.93)	3.24 (7.79)
	(2.12)	2.85 (11.7)	2.85 (11.6)	2.88 (10.5)
	OPTIMAL	1.94 (2.88)	1.93 (2.84)	1.85 (3.18)
N=300	(2.10)	2.77 (3.38)	2.80 (3.27)	2.74 (3.15)
	(2.12)	2.65 (4.14)	2.62 (4.32)	2.44 (4.05)
	OPTIMAL	2.05 (1.94)	2.05 (1.94)	2.06 (2.46)
N=500	(2.10)	2.56 (2.69)	2.56 (2.69)	2.70 (2.70)
	(2.12)	2.60 (2.84)	2.60 (2.84)	2.66 (2.73)

TABLE II (b) (FOR MODERATELY DECREASING λ, ZIPF'S LAW)

		T_λ		
		0.1	0.05	0.01
	OPTIMAL	8.90 (58.7)	10.4 (55.3)	6.98 (16.3)
N=100	(2.10)	9.65 (60.5)	12.9 (47.2)	8.08 (8.49)
	(2.12)	9.57 (48.8)	10.8 (44.0)	7.42 (21.4)
	OPTIMAL	9.00 (19.4)	10.4 (18.27)	6.90 (5.58)
N=300	(2.10)	9.26 (20.7)	11.1 (17.73)	8.00 (3.98)
	(2.12)	9.30 (16.7)	10.7 (15.26)	7.33 (8.26)
	OPTIMAL	9.07 (11.8)	10.4 (11.1)	7.06 (3.65)
N=500	(2.10)	9.36 (12.6)	11.1 (11.1)	7.66 (3.32)
	(2.12)	9.44 (10.5)	10.7 (9.56)	7.56 (5.01)

TABLE II(c) (CONSTANT λ)

		T_λ		
		0.1	0.05	0.01
	OPTIMAL	5.23 (43.1)	8.16 (62.2)	10.0 (1.00)
N=100	(2.10)	5.67 (46.0)	11.7 (58.8)	10.0 (5.54)
	(2.12)	5.61 (50.1)	8.82 (53.7)	10.4 (28.9)
	OPTIMAL	4.98 (13.9)	7.99 (20.4)	10.0 (1.00)
N=300	(2.10)	5.43 (15.7)	8.92 (21.8)	10.0 (2.52)
	(2.12)	5.46 (17.0)	8.45 (18.8)	10.4 (10.4)
	OPTIMAL	5.18 (8.88)	8.14 (12.8)	10.0 (1.00)
N=500	(2.10)	5.66 (9.62)	8.44 (13.5)	10.2 (3.03)
	(2.12)	5.83 (10.6)	8.46 (11.6)	10.5 (6.54)

TABLE II(d) (RANDOM λ)

		T_λ		
		0.1	0.05	0.01
	OPTIMAL	6.51 (46.9)	8.49 (57.5)	9.66 (19.1)
N=100	(2.10)	6.92 (46.8)	10.8 (53.5)	9.86 (6.11)
	(2.12)	6.93 (47.8)	9.09 (49.5)	9.77 (27.9)
	OPTIMAL	6.56 (15.7)	8.46 (19.2)	9.65 (6.41)
N=300	(2.10)	6.94 (16.0)	9.08 (19.3)	9.81 (2.64)
	(2.12)	6.98 (16.3)	8.96 (17.2)	9.80 (10.0)
	OPTIMAL	6.52 (9.18)	8.41 (11.3)	9.61 (3.79)
N=500	(2.10)	6.80 (9.83)	8.72 (12.1)	9.73 (3.46)
	(2.12)	6.84 (10.1)	8.67 (10.7)	9.83 (6.54)