Reliability-Estimation & Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs

Mark C. K. Yang
University of Florida, Gainesville
Anne Chao
National Tsing Hua University, Hsin Chu

Key Words — **Software reliability, software testing, stopping rules in testing**

Summary & Conclusions — **Software-testing (debugging) is one of the most important components in software development. An important question in the debugging process is, when to stop. The choice is usually based on one of two decision criteria:**

1. **when the reliability has reached a given threshold,**
2. **when the gain in reliability cannot justify the testing cost.**

Various stopping rules and software reliability models are compared by their ability to deal with these two criteria. Two new stopping rules, initiated by theoretical study of the optimal stopping rule based on cost, are more stable than other rules for a large variety of bug structures. The 1-step-ahead stopping rules based on the Musa *et al* basic execution and logarithmic Poisson models, as well as the stopping rule by Dalal & Mallows (1990), work well for software with many relatively small bugs (bugs with very low occurrence rates). The comparison was done by simulation.

1. INTRODUCTION

Acronyms[1]

LP	Musa *et al* logarithmic Poisson model
SR	stopping rule.

Software testing, or debugging, is one of the most important components in software development. In many projects, the time used in debugging can be around 50% of the total development effort [17]. An important question is when to stop the debugging process. Two obvious criteria are:

1. when the reliability has reached a given threshold,
2. when the gain in reliability cannot justify the testing cost.

To use criterion #1, we need to predict the remaining (future) failure rate of the software if testing is stopped. To use criterion #2, in addition to the prediction requirement, we need to identify a monetary cost equivalent for failures encountered by the user. Cost estimation is presented in Sherer [23] and Boehm [2]. This paper assumes that the monetary equivalent of future failure has been established.

[1]The singular & plural of an acronym are always spelled the same.

Prediction is always risky, especially when humans are involved. There are models that count the lines of code to predict the number of bugs in a software program (Lipow [12] and Brown, Maghsoodloo, Deason [4]). However, a well accepted formula has not been found, and it is unlikely that one bug-prediction formula, without referring to the nature of the software and the programmer skills, can cover the large variation in software development. A *bug* represents a defect component that will cause some type of failure or incorrect output with certain inputs. Sometimes *fault* or *error* is used for the term *bug*. The definitions can be different, but it is difficult to make them precise. They are approximations to a similar phenomenon and cause no difference in the mathematical aspect of software reliability.

Reliability prediction usually relies on the testing history of the software. In order to predict the future from the past, one needs a model to connect them. But so far, there is no simple way to identify a correct model for this connection. Nevertheless, some of the models fit a particular piece of software very well. When all the bugs are assumed to be the same size (measured by the failure rate), it becomes the Musa *et al* basic model [16] or the Goel-Okumoto model [9]. This same model is also known as the Jelinski & Moranda [11] model in statistical literature. When the bugs are not assumed to be the same size, a structure of the bug failure rates can be modeled, *eg*, the Littlewood-Verrall model [13, 14], in which a Gamma-prior is assigned to the failure distribution. So far, neither the equal size nor the Gamma-prior models have any theoretical justification. Thus, some researchers would rather fit the testing history (total faults encountered *vs* time) directly by reasonable curves. Among these empirical fits are the Musa *et al* logarithmic Poisson (LP) model [16], the Yamada *et al* Weibull model [24], and many other models (see [16: chapter 11]).

Once a model is preassigned, one can predict the future reliability if there are enough past data to estimate the parameters in the model. However, the choice of the best model usually fluctuates as observations continue. Thus, it is of both theoretical & practical interest to develop reliability estimation methods and SR that do not depend on a particular model or curve. This paper constructs two SR without assuming any distribution of the bug sizes. The idea is based on the similarity between the undiscovered bugs in software and the undiscovered species in an ecological system. This analogy has been used by Nayak [18] who proposed a recapture debugging process to get extra information for estimating the total number of bugs and for constructing a SR. While the Nayak procedure still depends on the equal failure rate assumption, our new SR are very general. Moreover, our new methods depend less on the recapture requirement. Here, *recapture* means that a bug can be observed more than once in the testing process. If a software bug, once discovered, is removed immediately, recapture debugging

0018-9529/95/$4.00

becomes impossible. But in many testing schemes, testing & debugging are not performed simultaneously. Thus, in the testing phase, the same bug can cause several failures, *ie*, the same bug can be observed repeatedly. Some recent developments in software engineering also advocate the separation of code writing, testing, and debugging. This method, usually referred to as the cleanroom procedure, has been shown to be better than the traditional simultaneous writing, testing, and debugging scheme in many small software development experiments; see Mills, Dyer, Linger [15] and Selby, Basili, Baker [22]. Recapture debugging also occurs when the software is already released to the public or to a group of trial customers. None of the traditional testing models can use the information of the repeated bug appearances. It seems now necessary to investigate the merit of recapture debugging in software testing.

The testing cases in the cleanroom procedure are chosen as a random sample for possible future inputs. Testing which uses this type of input is usually referred to as random testing. Its advantages & limitations have been discussed by Duran & Ntafos [8]. This is the only type of testing considered in this paper.

Nomenclature

- *Encounter*. A bug is *encountered* (in a *case* in a *round*) iff it: 1) causes a failure in that *case* in that *round*, and 2) is identified as the cause of that failure.
- *Case*. A single test of the software under specified conditions.
- *Round*. A set of a specified number of contiguous *cases*.
- *Singleton*. A bug that has been encountered exactly once in a specified period.
- *Doubleton*. A bug that has been encountered exactly twice in a specified period.
- *Test size*. Total number of *cases* tested.

Notation

S_n, B_n	number of [singletons, doubletons] discovered during rounds 1 through n inclusive
s_n, b_n	number of [singletons, doubletons] discovered during round n
$\hat{\ }$	implies an estimate
c_1	cost of testing 1 *case*
c_2	cost to the software producer for 1 failure after the release of the software
c	$M \cdot c_2/c_1$
$\mathfrak{I}(\cdot)$	indicator function: $\mathfrak{I}(\text{True})=1$, $\mathfrak{I}(\text{False})=0$
M	s-expected number of *cases* to be used by the costumers
m	initial number of bugs in a software
N	number of *cases* in 1 *round*; the stopping decision is made only at the end of each *round*, not during the *round*
n	implies *round #n*
T_q, R	[initial, remaining] bug-*encounter*-probability of the software
$X_i(n)$	number of times that bug i is *encountered* at the end of round n
q_i, p_i	[*encounter*, non-*encounter*] probability (per *case*) for bug i, $i=1,2...m$; $p_i+q_i \equiv 1$
N_0	initial test size

$\sum_i \quad \sum_{i=1}^{m}$.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Assumptions

1. Testing is performed in *rounds*; in each *round*, equal numbers of *cases* are tested. Failures are recorded, but not investigated, during each *round*.
2. The q_i is a constant from *case*-to-*case* and from *round*-to-*round*. For section 2.1 only, the q_i are known.
3. For sections 2.2 & 2.3 only, there are an unknown number, m, of bugs, each with unknown q_i, in the software before testing.
4. The bugs act s-independently.
5. Non-*encountered* bugs are never removed. When bugs are removed, removal is perfect, *ie*, a) removal is certain, b) no new bugs are inserted, and c) the q_i for non-removed bugs are not affected in any way.
6. Either of the following 2 debugging models can be used.
   a. Recapture debugging model: *encountered* bugs are not removed until the release of the software.
   b. Usual debugging model: *encountered* bugs are removed after each round. ◄

A consequence of these assumptions is that the failures are modeled by a discrete-time non-homogeneous Poisson process with a step-function failure intensity.

2. COST LOWER-BOUND AND NEW STOPPING RULES

2.1 Perspective

Assumption #2, in more detailed mathematical terms, is:

$$q_1 \geq q_2 \geq \ldots \geq q_m > 0. \tag{1}$$

$$T_q = \sum_i q_i;$$

$R < T_q$ if & when bugs are removed.

Though there is little difference in measuring the "time" axis by execution duration or *cases*, we use *cases* as the "time" unit. Assumption #2, in (1), is very general and our first goal is to find the lower bound for testing cost under any SR. It is well known that the optimal SR depends on the prior knowledge of (1) [21]. For example, if we know that $m=0$, then any testing is a waste. Because we have argued previously that no modeling of (1) can be justified, assumption #2 states that all the q_i are known; but it takes testing to find these bugs and to remove them. This section shows that under assumption #2, the optimal

SR and its cost can be found. This cost is the lower bound. If the cost of any SR is close to this bound, then improvement becomes unnecessary for this SR. All three numbers, c_1, c_2, M, are necessary in order to balance the testing cost and failure penalty.

Appendix A shows that the optimal SR is to stop at the first n such that:

$$\sum_i q_i \cdot (1-p_i^N) \cdot \mathcal{I}(X_i(n)=0) \leq N/c. \tag{2}$$

There is no simple formula for the optimal cost, but it can be found easily by simulation based on the optimal SR.

2.2 Recapture Debugging (Assumptions #3 & #6a)

Since the q_i are unknown, the SR in (2) cannot be used in practice. But (2) tells us that if there were a good estimate of the l.h.s of the inequality, we could construct a close-to-optimal SR. Since the $q_i << 1$, a good approximation for (2) is:

$$\sum_i q_i^2 \cdot \mathcal{I}(X_i(n)=0) \leq 1/c. \tag{3}$$

Notation

$$\theta(n) \quad \mathrm{E}\left\{ \sum_i q_i^2 \cdot \mathcal{I}(X_i(n)=0) \right\}.$$

$$\theta(n) = \sum_i q_i^2 \cdot p_i^{n \cdot N}. \tag{4a}$$

It is known [3, 5]) that:

$$\mathrm{E}\left\{ \sum_i \mathcal{I}(X_i(n)=2) \right\} = \binom{n \cdot N}{2} \cdot \sum_i q_i^2 \cdot p_i^{n \cdot N-2}. \tag{4b}$$

Since $p_i^{n \cdot N-2} \approx p_i^{n \cdot N}$,

$$\hat{\theta} = \left[\sum_i \mathcal{I}(X_i(n)=2) \right] / \binom{n \cdot N}{2} \equiv B_n / \binom{n \cdot N}{2}.$$

Thus, a reasonable adaptive SR for recapture debugging is — Stop at the first n such that:

$$B_n / \binom{n \cdot N}{2} \leq 1/c. \tag{5}$$

2.3 Usual Debugging (Assumptions #3 & #6b)

The number of doubletons needs to be estimated. By the simple equality: B_n = (Bugs in B_{n-1} are not encountered in round n) + (Bugs in S_{n-1} are encountered exactly once in round n) + b_n, and by the maximum-likelihood principle, recursive estimates are:

$$\hat{B}_n = \hat{B}_{n-1} \cdot (1-2\nu)^N + \hat{S}_{n-1} \cdot (1-\nu)^{N-1} / (n-1) + b_n; \tag{6}$$

$$\hat{S}_n = \hat{S}_{n-1} \cdot (1-2\nu)^N + s_n; \tag{7}$$

$$\nu \equiv 1/[N \cdot (n-1)].$$

Thus, an adaptive SR for this model is:

Stop at the first n such that:

$$\hat{B}_n / \binom{n \cdot N}{2} \leq 1/c. \tag{8}$$

Eq (5) & (8) are our two adaptive SR.

2.4 General Discussion

Data with recapture information can also be used to estimate R. According to the undiscovered species theory [5], the maximum-likelihood estimates for R under the recapture-debugging SR and usual-debugging SR are, respectively:

$$\hat{R}_{\#6a} = S_n / (n \cdot N), \tag{9a}$$

$$\hat{R}_{\#6b} = \hat{S}_n / (n \cdot N). \tag{9b}$$

They are useful because R is also one of the main concerns in software testing.

Adaptive SR (5) & (8) cannot be as efficient as (3) due to the large gap in prior information on the q_i. For example, if we know that the software has no bugs, then by (2), the optimal SR is not to test it at all. But when this fact is unknown, it would take a lot of testing to find it out. However, the inferiority of adaptive SR (5) & (8) to SR (3) does not imply that they are inferior to any other practical SR, because all of them are in the same blind situation on the distribution of the q_i. Our new adaptive SR and the other old SR are compared in section 3 by a simulation study, but here we investigate the conditions under which the adaptive SR (5) should be close to (3). Adaptive SR (8) is not investigated here due to its mathematical complexity.

The difference between the l.h.s of (3) & (5) is:

$$\Delta(n) = \sum_i q_i^2 \mathcal{I}(X_i(n)=0) - B_n / \binom{n \cdot N}{2}.$$

Appendix B shows that:

$$\mathrm{E}\{\Delta(n)\} \approx 0, \tag{10a}$$

$$\mathrm{Var}\{\Delta(n)\} = \sum_i q_i^2 \cdot p_i^{n \cdot N} \cdot [q_i^2 + 1/\binom{n \cdot N}{2}], \tag{10b}$$

$$\mathrm{StdDev}\{\Delta(n)\} \equiv \sqrt{\mathrm{Var}\{\Delta(n)\}}. \tag{10c}$$

Since $\mathrm{Var}\{\Delta(n)\} \to 0$ as $n \to \infty$, this approximation is asymptotically s-consistent. For smaller n, we use the coefficient of variation,

$$\mathrm{CV}(n) \equiv \mathrm{StdDev}\{\Delta(n)\}/\theta(n), \tag{11}$$

to judge the goodness of the approximation. This criterion is reasonable because if the error is much smaller than the mean value, the error is not serious. Eq (11) is calculated from (4a) & (10b).

Since the decision rule is at time $\theta(n) \approx 1/c$, the c is in a desirable range if,

$$\mathrm{CV}(n) \leq 0.5, \text{ for } \theta(n) = 1/c. \tag{12}$$

Though this range cannot be computed without knowledge of the q_i, it can be easily checked for any given distribution of the q_i. As demonstrated in section 3, adaptive SR (5) is usually good for a large range of c.

3. COMPARISON OF STOPPING RULES

One of the most convincing methods to compare SR is to use them in real software testing and see which one saves the most money. This is obviously unrealistic, because it is unlikely that any company is willing to release multiple versions of a software product with different SR. Thus, we have to use simulation. There are few existing records on the distribution of bug q_i in software. Actually, if a bug is immediately removed from the software at its discovery during testing, its q_i cannot be estimated. The only record we can find is from the nine IBM products in Adams [1: table 2]. All of the distributions are similar. We used the following q_i distributions and parameter values therein.[2]

A. Rapidly decreasing q_i (in exponential rate)

$q_i = K \cdot \alpha^i$, for $\alpha \in (0,1)$; $\alpha = 0.8, 0.7, 0.5$;

B. Moderately decreasing q_i (Zipf's Law)

$q_i = K/(\delta + i)$, with a constant δ; $\delta = 0, 10, 50$;

C. Constant q_i: $q_i = K$;

D. Uniform q_i: $q_i = K \cdot U_i$, where U_i is a uniformly distributed r.v. in (0, 1);

E. The Adams data in table 1.

For A - D:

$T_q = 0.05, 0.01, 0.001$;

$c = 10^5, 10^6, 10^7$.

A & B were chosen because the Adams data falls between them, *ie*, the q_i in table 1 decrease faster than Zipf's law but slower than the exponential rate. Constant rate is included because many SR are based on this assumption, and the uniform rate is used following Ross [20] in his simulation.

[2] K is a normalization constant so that $T_q = \Sigma_i q_i$; where T_q *is given.*

TABLE 1
Failure Rate Distribution from the Adams Data [1: table 2]
[$m=8$, $\eta_i \equiv$ number of bugs in category i]

i	η_i	$q_i(10^{-6})$
1	1	10000
2	7	3200
3	16	1000
4	13	320
5	48	100
6	91	32
7	82	10
8	75	3.2

Many simulations were run. Since each method requires a minimum amount of data to apply its SR, N_0 was chosen as 300, 500, 1000; N was chosen as 100. The SR compared were Ross [20], Rasmussen & Starr [19], Nayak [18], Goudie [10], Dalal & Mallows [7], and the Musa basic-execution and LP models. The detailed formulation of these SR and several other SR were laid out in [25]. The ability of each SR to estimate the remaining R is also examined. Since we know the true remaining R in the simulation, the,

$$\epsilon \equiv |\hat{R} - R|, \tag{13}$$

can easily be computed.

1000 simulations were done in each of the possible parameter combinations. Tables 2 & 3 give typical outputs. Only five of those SR are included.

TABLE 2
Average Testing & Penalty Cost For Six SR
[$N=100$, $m=100$, $T_q=0.05$, $c=10^6$, $N_0=1000$
The body of the table gives: "estimated mean cost — including total-test and penalty" and "standard-error of the estimated mean"; the latter is in () with units of 10^2]

	q_i Distribution				
Method	A, $\alpha=0.7$	B, $\delta=0$	C	D	E
Optimal	29.3 (0.24)	104.2 (0.26)	81.7 (0.32)	84.5 (0.30)	118.3 (0.23)
Recapture (5)	36.0 (0.36)	111.5 (0.59)	84.6 (0.31)	88.5 (0.36)	124.5 (0.42)
Usual (8)	38.1 (0.29)	108.5 (0.48)	85.2 (0.36)	87.9 (0.36)	123.4 (0.24)
Dalal/Mallows	39.7 (0.50)	116.0 (0.28)	85.4 (0.30)	90.5 (0.28)	147.8 (0.21)
Musa Basic	36.1 (0.39)	107.3 (0.24)	85.0 (0.24)	88.2 (0.32)	130.6 (0.25)
Musa LP	34.9 (0.28)	120.1 (0.97)	148.4 (0.32)	128.5 (2.24)	126.0 (0.51)

All the other SR, when adapted to our cost structure, were no longer competitive. Since our cost structure was not studied in the Musa nor in the Dalal/Mallows modeling, we put our version of their SR in appendix C. The results here do not

TABLE 3
Average Error (ϵ) In $\hat{R}$ For Six SR
[$N=100$, $m=100$, $T_q=0.05$, $c=10^6$, $N_0=1000$
The body of the table gives: "$\epsilon \equiv |\hat{R}-R|$" and "standard-error of the ϵ"; the latter is in (), both with units of 10^{-3}]

	q_i Distribution				
Method	A, $\alpha=0.7$	B, $\delta=0$	C	D	E
Optimal	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)	0.00 (0.000)
Recapture (5)	1.39 (0.039)	1.23 (0.035)	0.90 (0.021)	1.03 (0.025)	1.54 (0.041)
Usual (8)	2.07 (0.075)	1.18 (0.034)	2.70 (0.037)	1.74 (0.032)	1.72 (0.033)
Dalal/Mallows	1.55 (0.052)	1.33 (0.045)	0.83 (0.022)	1.07 (0.030)	2.14 (0.049)
Musa Basic	4.78 (0.042)	2.03 (0.046)	1.11 (0.029)	1.51 (0.040)	2.91 (0.049)
Musa LP	0.92 (0.025)	2.70 (0.116)	5.08 (0.135)	3.95 (0.141)	2.03 (0.077)

necessarily imply that the other methods are less effective in the circumstances they were designed for. From the simulation results, we observed the following.

1. As we might anticipate, there is no definite winner in all the q_i configurations. The Musa LP model does extremely well in rapidly decreasing q_i, but does poorly in others. For Zipf's law, constant q_i, and uniform q_i, our two adaptive SR and the SR by Dalal/Mallows and Musa Basic models have similar performance. Taking the standard error of estimation into consideration, their differences are not s-significant. But if we consider the Adams data to be typical, then the our adaptive SR are better in average cost reduction and in estimation of R.

2. As mentioned in section 2, it is not fair to compare the optimal SR (3) with other SR, because there is a tremendous prior information gap between them. However, the simulation shows that for cases B, C, D, the existing SR are nearly optimal. Further research on improving the testing strategies under these three models does not seem worthwhile.

3. For the same T_q, it is easier to debug "one bug with large q_1" than "many bugs each with small q_i". Thus T_q has little effect on the total cost. Testing is more difficult with low-q_i bugs.

4. The range of c is important in determining the closeness between the adaptive and the optimal SR. The desirable ranges (12) for c, for each case A - E, are:

- A: none
- B: $3\cdot10^5$ to 10^7
- C: $6\cdot10^4$ to 10^6
- D: $5\cdot10^4$ to $6\cdot10^5$
- E: $2\cdot10^6$ to $1.4\cdot10^{10}$.

Tables 2 & 3 clearly show the relationship between the cost discrepancy from the optimal SR and the closeness of c to the desirable range. The lengthy desirable range for the Adams data seems to indicate that the adaptive SR will work well in commercial software.

5. We checked the robustness of the existing SR when the pure death assumption of bugs in debugging is slightly violated (see assumption 5). From a mathematical viewpoint, there is no difference between a hidden bug B behind a bug A, and a bug B generated during the correction of bug A. In other words, they share the same consequence; B appears only after A is removed. We simulated this situation by assuming a β birth rate when a bug is removed. The q_i of the newly generated bug is randomly picked from the original bug pool. We tried $\beta=0.05$ and 0.10; the results were very similar to those in tables 2 & 3.

APPENDIX A

Derivation of the Optimal Stopping Rules (2)

We found no elementary proof for (2). However, if we use the 1-step-ahead SR, then (2) is easy to derive. The 1-step-ahead SR is: "If the next step is not profitable, then stop, regardless of what would happen many steps ahead." To see this, suppose there are still k bugs with $\{q_i\}_{i=1}^k$ left in the software after round n. Then if we decide to test one more round, the cost is $N\cdot c_1$. The mean reduction of penalty is:

$$c_2\cdot M\cdot \mathrm{E}\left\{\sum_{i=1}^{k} q_i\cdot \Pr\{\text{bug } i \text{ is discovered in the nest } N \text{ tests}\}\right\}, \text{ or}$$

$$c_2\cdot M\cdot \sum_i q_i\cdot(1-p_i^N). \tag{A-1}$$

Thus, the 1-step-ahead SR says that we stop iff (A-1) $\le N\cdot c_1$ which is the same as (2).

For a rigorous proof, we follow [6: theorem 3.3].

Notation

$U(n)$	$\Sigma_i\, q_i\cdot \mathfrak{I}(X_i(n)=0)$
τ	any SR: (stop at τ)
ϕ	optimal SR for $\mathrm{E}\{C(\phi)\} \le \mathrm{E}\{C(\tau)\}$: stop at ϕ
$\mathfrak{F}_n$	sigma field generated by all the previous samples up to n.

If the program is released after test period n, the cost is:

$$C(n) = c_2\cdot M\cdot U(n) + c_1\cdot n\cdot N. \tag{A-2}$$

Our purpose is to find the ϕ for any τ. A SR is a decision that depends only on the sampling information from the past, not the future. To put it in the usual notation, τ is a r.v. such that the event $(\tau=n) \in \mathfrak{F}_n$. Without loss of generality, we let:

$$c_1=1 \text{ and } c_2\cdot M=c;$$

$$g(n) \equiv -c\cdot U(n) - n\cdot N.$$

According to [6: theorem 3.3], we need to show:

i. The set $A_n \equiv \{\mathrm{E}\{g(n+1)|\mathfrak{F}_n\} \le g(n)\}$ is monotonically increasing in n, and

ii. $\lim_n\{\inf\int_{\phi>n} g^+(n)\, dP\} = 0$,

in order for the SR ϕ, defined as: "stop at the first n such that $g(n) \geq E\{g(n+1)|\mathcal{F}_n\}$ (same as SR (2))", to be optimal among all the SR τ satisfying $\lim_n\{\inf\int_{\tau>n} g^-(n)\, dP\} = 0$. To show #i, note that $E\{g(n+1)|\mathcal{F}_n\} \leq g(n)$ implies:

$$-c\cdot \sum_{i:X_i(n)=0} q_i\cdot \Pr\{\mathcal{I}(X_i(n+1)=0)\,|\,X_i(n)=0\}$$

$$-\ (n+1)\cdot N \leq -c\cdot U(n) - n\cdot N$$

$$\Leftrightarrow\ c\cdot \sum_i q_i\cdot(1-p_i^N)\ \cdot\ \mathcal{I}(X_i(n)=0)\ \leq\ N$$

$$\Rightarrow\ c\cdot \sum_i (1-p_i^N)\cdot \mathcal{I}(X_i(n+1)=0)\ \leq\ N$$

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

Thus #i is proven. Because $g^+(n)=0$ for all n, #ii is true. Since,

$$U(n) \leq 1,$$

$$g^-(n) = c\cdot U(n) + n\cdot N < c + n\cdot N,$$

$$\int_{\tau>n} g^-(n)\, dP \leq (c+n\cdot N)\cdot \Pr\{r>n\} \to 0,$$

as $n \to \infty$, for $E\{\tau\} < \infty$.

Thus, we have shown that ϕ is the optimal SR for all SR with finite s-expectations. Suppose $E\{\tau\} = \infty$, then $E\{g(\tau)\} \leq -E\{\tau\} = -\infty$. Because the SR that stops with $\phi \equiv 1$ has an s-expectation greater than $-(c+N)$, this τ cannot be optimal. We have proven (2) is optimal among all τ.

APPENDIX B

Derivation of the Variance for (10)

Notation

Ψ_i $\quad q_i^2\cdot \mathcal{I}(X_i(n)=0) - B_n/\binom{n\cdot N}{2}$.

Since we have shown that $E\{\Delta(n)\} \approx 0$,

$$\mathrm{Var}\{\Delta(n)\} = E\left\{\sum_i \Psi_i\right\}^2$$

$$= \sum_i E\{\Psi_i\}^2 + \sum_{i,j\ (i\neq j)} E\{\Psi_i\cdot\Psi_j\}$$

By $E\{\mathcal{I}(X_j(n)=2)\} = \binom{n\cdot N}{2}\cdot q_i^2\cdot p_i^{n\cdot N-2}$,

we can show that the a) cross product terms ≈ 0, and b) squared terms become (10).

APPENDIX C

Derivation of Stopping Rules for The MUSA and DALAL/MALLOWS Models

C.1 Musa Model

Notation

T_i	time that bug i (in the order of new discovery) is discovered
m_t	total number of distinct bugs discovered in $[0,t]$
$'$	implies the derivative
$\mu(t)$	mean number of failures up to time t.

The time unit is one testing case.

The Musa basic-execution model contains two parameters, ν_0 & β_1, that relate the failure rate $\lambda(t)$ and current t by:

$$\lambda(t) = \nu_0\cdot\beta_1\cdot\exp(-\beta_1\cdot t). \tag{C-1}$$

Each time a new bug is discovered, $\hat{\nu}_0$ & $\hat{\beta}_1$ can be found by solving equation [6: (12.41) and then (12.40)]. The reduction in failure rate by one additional test of a unit of time is the derivative $\lambda'(t)$ of (C-1). Thus, the 1-step-ahead SR is to stop if,

$$-\lambda'(t) \leq 1/c. \tag{C-2}$$

It can be shown that:

$$-\lambda'(t) = (\nu_0 - \mu(t))\cdot\beta_1^2$$

Use $\hat{\nu}_0$ and $m_t = \hat{\mu}(t)$; (C-2) becomes,

$$(\hat{\nu}_0 - m_t)\cdot\hat{\beta}_1^2 \leq 1/c.$$

The projected R is $\hat{\beta}_1\cdot(\hat{\nu}_0 - m_t)$.

The LP model contains the two parameters, β_0 & β_1, that relate the failure rate to the current time t by:

$$\lambda(t) = \beta_0\cdot\beta_1/(\beta_1\cdot t+1).$$

The two parameters can be estimated from the failure data by solving [16: (12.57) and (12.55)]. As in the basic model, an intuitive SR, using $\lambda'(t)$, is to stop when,

$$\hat{\beta}_0\cdot\hat{\beta}_1^2\cdot\exp(-2m_t/\hat{\beta}_0) \leq 1/c.$$

The R is estimated by:

$$\hat{\beta}_0\cdot\hat{\beta}_1\cdot\exp(-m_t/\hat{\beta}_0).$$

Remark

The SR derived for these two models are the 1-step-ahead SR. They might not be the optimal SR for these models. The difficulty of deriving a practical optimal SR for a given model

is shown in Singpurwalla [24] where only a 2-step testing scheme is considered. ◄

C.2 Dalal/Mallows [7]

Notation

$S(t)$ $\sum_{i=1}^{m_t} T_i$

μ solution of (C-3)

ξ cost of an unremoved bug.

$$\mu(\exp(\mu \cdot t) - 1)/(\exp(\mu \cdot t) - 1 - \mu \cdot t) = m_t/S(t). \quad \text{(C-3)}$$

Their SR is to stop when,

$$m_t \leq (\exp(\mu \cdot t) - 1) \cdot c_1/(\xi \cdot \mu)$$

The ξ has no direct correspondence in our cost structure. We adjust the c_1/ξ ratio so that is has the best performance. The R is estimated as:

$$\mu \cdot m_t/(\exp(\mu \cdot t) - 1).$$

REFERENCES

[1] E.N. Adams, "Optimizing preventive service of software product", *IBM J. Research & Development*, 1984 Feb, pp 2-14.

[2] B.W. Boehm, "Software engineering economics", *IEEE Trans. Software Eng'g*, vol SE-10, 1984 Jan, pp 4-12.

[3] P.K. Banerjee, B.K. Sinha, "Optimal and adaptive strategies in discovering new species", *Sequential Analysis*, vol 4, 1985, pp 111-112.

[4] D.B. Brown, S. Maghsoodloo, W.H. Deason, "A cost model for determining the optimal number of software testing cases", *IEEE Trans. Software Eng'g*, vol 15, 1989 Feb, pp 218-221.

[5] A. Chao, "On estimating the probability of discovering a new species", *Annals of Statistics*, vol 9, 1981, pp 1339-1342.

[6] Y.S. Chow, H. Robbins, Siegmund, *The Theory of Optimal Stopping*, 1991; Dover Publications.

[7] S.R. Dalal, C.L. Mallows, "Some graphical aids for deciding when to stop testing software", *IEEE J. Selected Areas in Communications*, vol 8, 1990 Feb, pp 169-175.

[8] J.M. Duran, S.C. Ntafos, "An evaluation of random testing", *IEEE Trans. Software Eng'g*, vol SE-10, 1984 Jul, pp 438-444.

[9] A.L. Goel, K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance resources", *IEEE Trans. Reliability*, vol R-33, 1984 Jun, pp 176-183.

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

[11] Z. Jelinski, P.B. Moranda, "Software reliability research", *Statistical Computer Performance Evaluation* (W. Freiberger, *Ed*), 1972, pp 465-484.

[12] M. Lipow, "Number of faults per line of code", *IEEE Trans. Software Eng'g*, vol SE-8, 1982 Jul, pp 437-439.

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

[14] B. Littlewood, J.L. Verral, "A Bayesian reliability growth model for estimating software reliability", *J. Royal Statistical Soc.* C, 1973, pp 332-346.

[15] H.D. Mills, M. Dyer, R.C. Linger, "Cleanroom software engineering", *IEEE Trans. Software Eng'g*, vol SE-13, 1987 Sep, pp 19-25.

[16] J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability Measurement, Prediction, Application*, 1987; McGraw-Hill.

[17] G.J. Myers, *Software Reliability Principles & Practices*, 1976; John Wiley & Sons.

[18] T.K. Nayak, "Estimating population size by recapture sampling", *Biometrika*, vol 75, 1988, pp 113-20.

[19] S.L. Rasmussen, N. Starr, "Optimal and adaptive stopping in the searching for new species", *J. Amer. Statistical Assoc.*, vol 74, 1979 Sep, pp 661-667.

[20] S.N. Ross, "Software reliability: The stopping rule problem", *IEEE Trans. Software Eng'g*, vol SE-11, 1985 Dec, pp 1472-1476.

[21] N.D. Singpurwalla, "Determining an optimal time interval for testing and debugging software", *IEEE Trans. Software Eng'g*, vol 17, 1991 Apr, pp 313-319.

[22] R.W. Selby, V.R. Basili, T.T. Baker, "Cleanroom software development: An empirical evaluation", *IEEE Trans. Software Eng'g*, vol SE-13, 1987 Sep, pp 1027-1037.

[23] S.A. Sherer, "Measuring the risk of software failure: A financial application", *Proc. 10th Int'l Conf. Information Systems*, 1987, pp 237-245.

[24] S. Yamada, J. Hishitani, S. Osaki, "Software-reliability growth with a Weibull test-effort: A model & application", *IEEE Trans. Reliability*, vol 42, 1993 Mar, pp 100-105.

[25] M.C.K. Yang, A. Chao, "Comparisons of methods in reliability estimation and stopping rules for software testing", *SERC Technical Report*, SERC-TR-58-F, SERC, 1992 Apr; Univ. of Florida.

AUTHORS

Dr. Mark C.K. Yang; Dept. of Statistics; Univ. of Florida; Gainesville, Florida 32611 USA.

Internet (e-mail): yang@stat.ufl.edu

Mark C.K. Yang received the BS (1964) in Electrical Engineering from National Taiwan University, MS (1967) in Mathematics and PhD (1970) in Statistics from the University of Wisconsin, Madison. He has worked with the US Department of Energy, US NASA, Bell Labs, and US Naval Research Lab. He has published more than 50 research articles in refereed statistical & engineering journals and is a professor at the University of Florida. His research interests are sampling, optimal stopping rules, and reliability.

Dr. Anne Chao; Inst. of Statistics; National Tsing-Hua University; Hsin Chu, TAIWAN.

Internet (e-mail): nthut060@twnhoe10.edu.tw

Anne Chao received the BS (1973) in Mathematics from National Tsing-Hua University and PhD (1977) in Statistics from the University of Wisconsin, Madison. She has published research papers in *Annals of Statistics*, *Biometrics*, *Biometrika*, *J. Amer. Statistical Assoc*, *IIE Trans.*, and other journals. She is now a professor at the National Tsing Hue University. Her research interests are wildlife sampling, coverage probability, and software reliability.

Manuscript received 1994 March 16.

IEEE Log Number 94-10822 ◄TR►