The Effect Of Imperfect Error Detection on Reliability Assessment Via Life Testing

Paul E. Ammann
Susan S. Brilliant
John C. Knight

Computer Science Technical Report No. TR-92-16
May 29, 1992

Submitted to *IEEE Transactions on Software Engineering*.

The Effect Of Imperfect Error Detection On Reliability Assessment Via Life Testing

Submitted To IEEE Transactions On Software Engineering
February, 1992

Paul E. Ammann[1, 2]
Department of Information And Software Systems Engineering
George Mason University
Fairfax, VA 22030

Susan S. Brilliant[3]
Department of Mathematical Sciences
Virginia Commonwealth University
Richmond, VA 23284

John C. Knight[2] †
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

Abstract

Measurement of software reliability by life testing involves executing the software on large numbers of test cases and recording the results. The number of failures observed is used to bound the failure probability even if the number of failures observed is zero. Most analyses assume that all failures will be observed but in practice this will rarely be the case. In this paper we examine the effect of imperfect error detection, *i.e.*, the situation in which a failure of the software may not be observed. If the conventional analysis associated with life testing is used, the confidence in the bound on the failure probability is optimistic. Our results show that imperfect error detection does not necessarily limit the ability of life testing to bound the probability of failure to the very low values required in critical systems. However, we show that the confidence level associated with a bound on failure probability cannot necessarily be made as high as desired unless very strong assumptions are made about the error detection mechanism. Such assumptions are unlikely to be met in practice, and so life testing is likely to be useful only for situations where very high confidence levels are not required.

Index Terms: Error Detection, Software Reliability Assessment, Software Testing, Test Oracles.

---

[1] This work supported in part by the National Science Foundation under grant CCR-90-10036.

[2] This work supported in part by NASA under grant NAG_1-1123-FDP.

[3] This work supported in part by the National Science Foundation under grant CCR-90-96128.

† Point of contact: email knight@virginia.edu, phone (804) 982-2216

1. INTRODUCTION

An important step in the development of many computing systems is some form of reliability assessment. In some cases it is essential to have confidence that the system achieves a required level of reliability before it is put into service. It might be necessary, for example, to demonstrate that a safety-critical system achieves the level of reliability prescribed by a regulatory agency (*e.g.* [3]).

Reliability can either be estimated from observed behavior or predicted. Estimation can be achieved by a process known as *life testing*, in which the long-term behavior of the product is observed and its reliability is estimated from the observations. Life testing is effected by operating the product as it would be in service. Inputs are supplied from the operational distribution and outputs consumed as they would be in the target environment. A single "use" of the product under such circumstances is usually referred to as a test, and the number of failures observed (usually zero) over an extended period can be used to provide an estimate of the failure rate per unit time. The estimate takes the form of a confidence interval. The reliability is shown to be no worse than some specific bound with a certain confidence. The probability that the reliability is worse than the computed bound is determined by the confidence level.

The alternative to direct measurement is prediction. Prediction of the reliability of hardware systems is usually based on a model. Many models use Markov methods to analyze the effects of component failures (and possibly component repair) [11]. The system's overall reliability is estimated by the model based on reliability data of the individual components. The data for individual components is obtained either by life testing or from models of the components themselves. This procedure is followed because it allows reasonable estimates to be computed much more quickly than would be possible with system-level life testing.

Software reliability has proved much harder to predict because hardware-like Markov models cannot be used. Some progress has been made with reliability-growth models [9] but such models are of limited applicability because they depend upon assumptions that are not always valid. Life testing at the system level is therefore attractive as an assessment technology, and has been advocated by many authors [8, 10, 12].

Life testing of software has been criticized by Miller [6, 7] and by Butler and Finelli [2] in a number of ways. Miller points out, for example, that it is difficult to ensure that the distribution from which the test cases are selected is identical to the operational distribution. He also observes that the lower the probability of failure that must be demonstrated, the more tests are required in the life testing procedure. The amount of testing required for assessing the reliability of safety-critical systems is, in general, infeasible.

It might be possible to overcome such difficulties in some situations. For example, the large number of tests required might be executed in a reasonable amount of time by a fast computer or many computers operating in parallel. However, in this paper we examine a different problem with life testing of software. We refer to this difficulty as *imperfect error detection.* Error detection is the process by which it is determined that a software system has failed. The analysis used in life testing to estimate the reliability of the software is based on knowledge of the outcome of some specified number of tests. The traditional analysis assumes the existence of an *oracle,* an observer of tests that can determine with absolute certainty whether the software operates correctly.

Weyuker labels a program *non-testable* if no oracle exists for the program [13]. She defines a *pseudo-oracle,* which might be implemented by an independently written program designed to satisfy the same specification as the non-testable program, and discusses testing "non-testable"

programs using a pseudo-oracle. A pseudo-oracle is subject to two types of errors. It may report a failure when the software is correct, or it may not report an actual failure. The former leads to a fruitless effort to find a nonexistent deficiency in the software, but has no bearing on the subsequent reliability analysis. The latter, however, goes unnoticed and leads to a reliability analysis based on incorrect data.

A premise of this paper is that most programs are "non-testable" by Weyuker's definition, and thus whatever error detection mechanism is used is likely to exhibit pseudo-oracle behavior. In this paper we use the term pseudo-oracle broadly to refer to any error detector that is not known to be perfect. Weyuker's paper identifies the qualitative result that pseudo-oracles may be unreliable; in this paper we attempt to quantify the extent of the problem. We examine the effects of imperfect error detection and show how to compute realistic reliability bounds in the face of uncertainty about the quality of the pseudo-oracle.

In section 2 we present a model for testing with imperfect error detection. We analyze the model for the case in which the quality of the pseudo-oracle is known in section 3. Then, in section 4, we extend the analysis to the more realistic case in which the behavior of the pseudo-oracle is known only in a probabilistic sense. Finally, in section 5 we present our conclusions.

2. ERROR DETECTION

The goal of life testing is to estimate the reliability of a program P. The reliability parameter for which we will obtain a confidence interval is the failure probability of P, which we denote p. We observe P's behavior on inputs selected at random from an expected usage distribution, consult a pseudo-oracle to evaluate the output of P, and record the decision. We need a measure of the quality of the pseudo-oracle in order to incorporate its effects into our

reliability analysis. We define the *hidden failure probability*, denoted r, to be the conditional probability that the pseudo-oracle does not reject P's output, given that P has failed. The hidden failure probability reflects the ability of the pseudo-oracle to detect the failures of the particular implementation being tested; its value refects the relationship between the failure behaviors of the implementation and of the pseudo-oracle.

In order to make our analysis tractable and to focus on the issues of greatest concern, we restrict our attention in two ways:

(1) We assume that the pseudo-oracle is not repaired. For the few test cases in which the output of the program is incorrectly rejected, we assume that the output of the program is validated by some unspecified means (perhaps by hand).

(2) We do not model repair of the program. Instead we assume that, if the oracle correctly rejects the output of the program on a particular case, the reliability assessment procedure starts over from the beginning after the program is repaired. Because, in general, fixing a fault can have an arbitrary effect on the functionality of a program, restarting after repair is an essential part of the reliability estimation process. Thus this assumption will be satisfied if reliability estimation is done properly.

The assessment procedure described above can be represented by the two-state Markov model shown in figure 1. The states in the model represent the tester's knowledge about the existence of a fault in P. Life testing begins in the initial state, in which P contains no known flaws. Recall that r, the hidden failure probability, measures the conditional probability that the pseudo-oracle fails, given that P has failed. Thus the conditional probability that a failure will be reported, given that one has occurred, is $1-r$, and the unconditional probability that a failure will

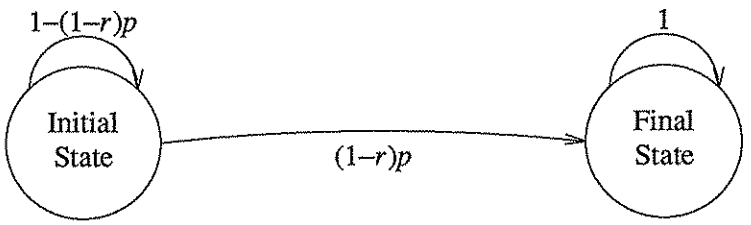


Fig. 1. Two State Markov Model For Reliability Assessment

be observed is $(1-r)p$. We move to the final state when a failure is observed; otherwise, with probability $1-(1-r)p$, we remain in the initial state. Entering the final state indicates that P contains a known fault that must be repaired. After the repair, the reliability assessment process begins anew in the initial state, with new values for both of the parameters p and r.

We begin our analysis by considering the effects of a faulty pseudo-oracle on the number of tests that must be executed before the first failure is detected. We denote this number of tests by the random variable T. Each test case is a Bernoulli trial, with probability of detected failure (*i.e.* moving to the final state in figure 1) of $(1-r)p$. Examining the value of T in isolation does not reveal the extent to which its distribution is influenced by imperfect error detection. As a basis for comparison, therefore, we also analyze the behavior of an oracle. We adopt the convention of marking measures on oracle-based testing with a tilde. Thus, for example, we define the random variable $\tilde{T}$ to denote the number of test cases that are executed before the first failure is detected by an oracle. $\tilde{T}$'s distribution, like T's, is geometric. The probability that an oracle will reveal a failure on a single test case is just p, the probability that the failure occurs, so the parameter of $\tilde{T}$'s distribution is p.

Now consider the goal of the reliability assessment process. We wish to bound the failure probability of the program to a particular value with a given statistical level of confidence. For example, we may wish to ensure that the failure probability is less than or equal to 10^{-4} with confidence 0.99. The interpretation of this statement is that if a large number of programs are assessed to have a failure probability of less than 10^{-4}, then on average no more than 1% of them will have failure probabilities that are greater than 10^{-4}.

For any specific assessment, both the bound for p and the confidence level C will usually be set by external requirements. The key issue that remains is to determine the number of successful tests required to show that the software system being assessed meets the prescribed bound with the required confidence. In general, we need to execute some number of tests, U, such that

$$\text{prob}(T \leq U) = C$$

Recall that the distribution of the random variable T, the number of tests needed to reveal a failure, is geometric with parameter $(1-r)p$, so the number of tests needed will depend upon both p and r.

To find the value of U that gives us the desired confidence interval, we solve the above equation using the desired bound on the failure probability as the parameter p in the probability distribution for T, and the desired confidence level as the value of C. The equation means that if we execute U tests and the actual failure probability is p, we will observe a failure with probability C. Thus, the execution of U tests without observing a failure gives us confidence C that the actual failure probability is bounded by p.

Before we attempt to solve the above equation, we consider its counterpart for an oracle. The distribution for $\tilde{T}$ is geometric with parameter p, so the probability that $\tilde{T}$ assumes any specific value t is given by:

$$\text{prob}(\tilde{T}=t) = p(1-p)^{t-1} \ , \ t=1,2,...$$

Thus

$$\text{prob}(\tilde{T}\leq\tilde{U}) = \sum_{t=1}^{\tilde{U}} p(1-p)^{t-1}$$

so the equation giving the number of tests $\tilde{U}$ needed to give the desired confidence interval using an oracle is

$$\sum_{t=1}^{\tilde{U}} p(1-p)^{t-1} = C$$

We can solve this equation for $\tilde{U}$:

$$p\left[\frac{1-(1-p)^{\tilde{U}}}{1-(1-p)}\right] = C$$

$$(1-p)^{\tilde{U}} = 1-C$$

$$\tilde{U} = \frac{\ln(1-C)}{\ln(1-p)}$$

Thus, if an oracle is available, this last formula is immediately useful for assessing P. The number of tests, $\tilde{U}$, that must be executed without observing a failure to give any desired bound on failure probability and confidence level can be computed by substituting the target values for p and C in the formula.

3. ANALYSIS FOR KNOWN HIDDEN FAILURE PROBABILITY

The analysis of the previous section was undertaken to permit a comparison between testing using pseudo-oracles, which are generally available, and using oracles, which are not

available, but are usually assumed to be in most analysis of life testing results. In this section we begin our analysis of testing using a pseudo-oracle. Initially we assume that the hidden failure probability, r, is known. In the next section we relax this assumption.

Recall that we want to calculate U, the number of successful tests needed to obtain the desired confidence interval. Also recall that T, like $\tilde{T}$, has a geometric distribution, but that the parameter of the distribution is $(1-r)p$ rather than p. The derivation of U, then, parallels the derivation of $\tilde{U}$, with each occurrence of p in the analysis of the previous section replaced by $(1-r)p$. The analysis yields the following expression for U:

$$U = \frac{\ln(1-C)}{\ln(1-(1-r)p)}$$

The above formula gives the number of tests that must be executed without an error report from the pseudo-oracle. The effects on U of changing the bound for p or the confidence level C are not immediately clear from this expression. These effects are very important in practice, not only in performing life testing analysis but also in setting the parameters to be used. The influence of the failure probability bound is illustrated by figure 2. Figure 2 shows U as a function of r for three different values of the failure probability bound p, namely $p = 10^{-3}$, 10^{-4}, and 10^{-5}. In this figure, the confidence level C is held constant at the value 0.99, and r is varied from 0 to 1. The figure shows that for each factor that the bound for p is decreased, U is increased by the same factor. For example, if achieving a bound of p requires U tests, then achieving a bound of $p/2$ requires $2U$ tests. In addition, reducing the visibility of the failure domain by a given factor has the effect of increasing the number of tests required by the same factor to maintain the same bound on the probability of failure. Thus, if achieving a bound of p for a given $1-r$ requires U tests, then achieving the same bound p for $\frac{(1-r)}{2}$ requires $2U$ tests.

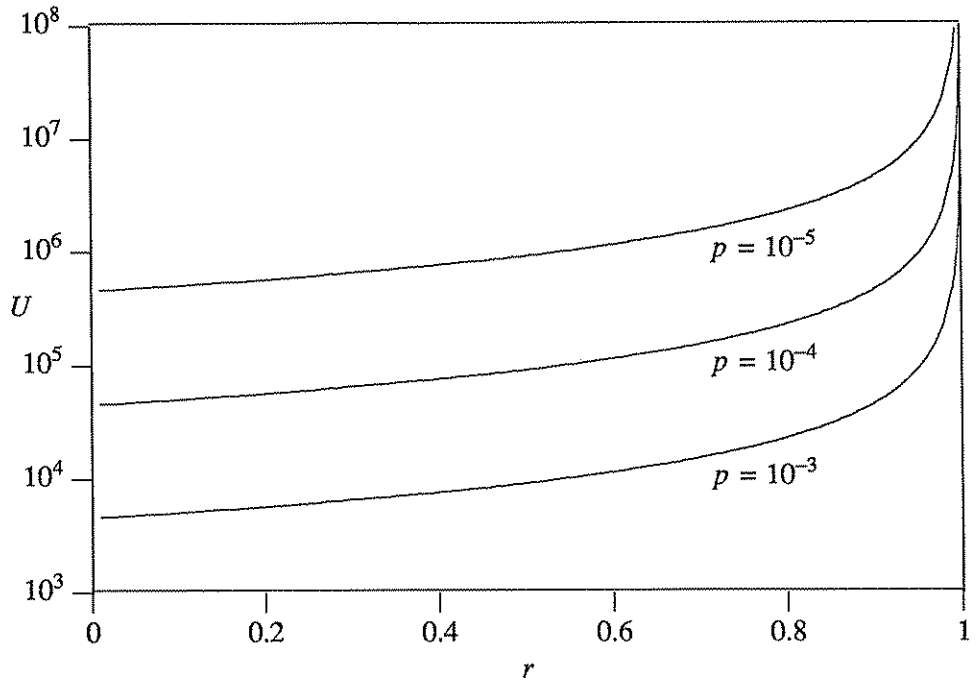


Fig. 2. U **vs.** r **for** $p = 10^{-3}, 10^{-4}, 10^{-5}, C = 0.99$

The effect of changing the confidence level is addressed in figure 3. Figure 3 again shows U as a function of r, but this time for three different values of the confidence level C, namely $C = 0.9, 0.99$, and 0.999. In figure 3, the bound for p is held constant at the value 10^{-4} and again r is varied from 0 to 1. The most striking observation from figure 3 is that a dramatic increase in the confidence level, C, results in only a small increase in the number of tests required, U. For example, increasing from C from 0.9 to 0.999 requires increasing the number of tests, U, by less than one order of magnitude.

To isolate the effect of imperfect error detection on the necessary number of tests, consider the ratio:

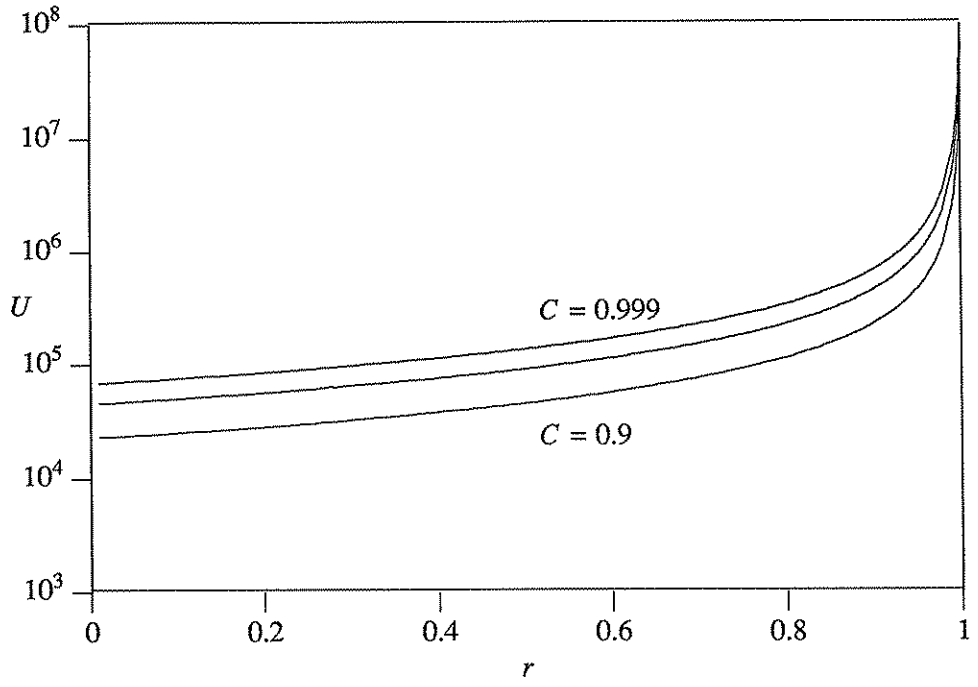


Fig. 3. U **vs.** r **for** $C = 0.9, 0.99, 0.999$, $p = 10^{-4}$

$$\frac{U}{\tilde{U}} = \frac{\frac{\ln(1-C)}{\ln(1-p+rp)}}{\frac{\ln(1-C)}{\ln(1-p)}} = \frac{\ln(1-p)}{\ln(1-p+rp)}$$

The values of p that we wish to obtain as a bound will be small. Below we find an approximation for $\frac{U}{\tilde{U}}$ for typically small values of p. The approximation depends on the fact that, for x close to one, $\ln(x) \approx x-1$. For small values of p, the quantities $(1-p)$ and $(1-p+rp)$ are both very close to one, Thus:

$$\frac{U}{\tilde{U}} = \frac{\ln(1-p)}{\ln(1-p+rp)} \approx \frac{(1-p)-1}{(1-p+rp)-1} = \frac{1}{1-r}$$

To a first approximation, then, $\frac{U}{\tilde{U}}$ is independent of C and nearly independent of p. Thus the factor by which the number of tests must be increased to compensate for the fallibility of the error detector depends almost entirely on the hidden failure ratio r.

In figure 4 we show $\frac{U}{\tilde{U}}$, the factor by which the required number of tests increases, as a function of r, the probability that a failure is hidden by the oracle.

Although the exact value of $\frac{U}{\tilde{U}}$ is a function of p, the approximation given above is excellent for

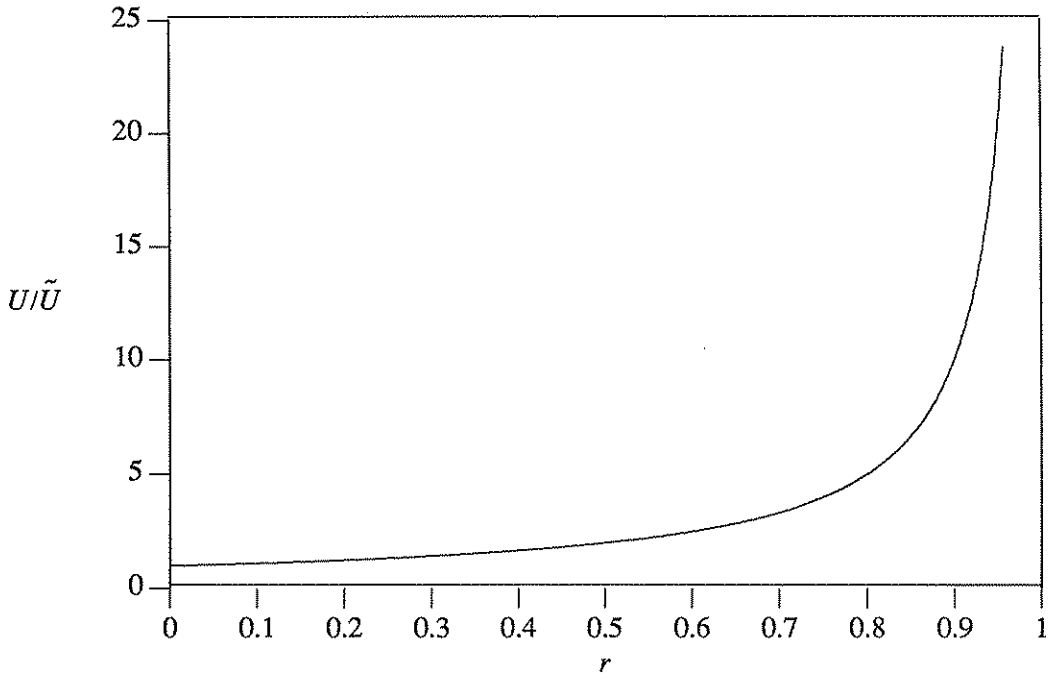


Fig. 4. $U/\tilde{U}$ vs. r for small p.

the small values of p typically used in life testing. For example, the graph of $\frac{U}{\tilde{U}}$ with $p = 10^{-3}$ is indistinguishable from that of the approximation $(\frac{1}{1-r})$ at the resolution provided in figure 4.

Figure 4 reveals that the pseudo-oracle performs nearly as well as an oracle for even moderate values of r. However, as r approaches 1, the relative performance of the pseudo-oracle deteriorates rapidly.

4. ANALYSIS FOR UNKNOWN HIDDEN FAILURE PROBABILITY

If the specific value of the hidden failure probability were known for a given pseudo-oracle and program combination, it would be a simple matter to consult figure 4 to determine how many more tests to run to compensate for the imperfection of the pseudo-oracle. For example, if it were known that $r = 0.8$, then reliability evaluation with this pseudo-oracle would require about five times as many tests as with an oracle to achieve the same bound for the failure probability p with the same confidence C. If r were known to be 1, it would of course be impossible to obtain any reliability bound.

Unfortunately, the value of the parameter r is never available in practice, and we do not learn more about r as testing proceeds. The variability of r is likely to be large. It is reasonable to expect that many systems have values of r that are very low, and perhaps zero. However, other systems will have quite large r values, and there is no way to recognize these systems during testing. Because the value of r is unknown, any estimate or bound computed using life testing results and based on $\tilde{U}$ could be arbitrarily inaccurate.

We can regard a particular program and pseudo-oracle combination as having been drawn from a population of systems having a particular distribution of r values. Below we present an

analysis that incorporates information about the distribution for r into the life testing analysis.

The probability that U tests will reveal a failure of a program having failure probability p, given that the hidden failure probability is r is

$$\text{prob}(T \le U \mid r) = \sum_{t=1}^{U} (1-r)p\ (1-(1-r)p)^{t-1}$$

In other words, this quantity represents the conditional probability that U tests reveal a failure, given a particular value for r. What we want is the unconditional probability that U tests reveal a failure, given by:

$$\text{prob}(T \le U) = \sum_{i} f_i \sum_{t=1}^{U} (1-r_i)p\ (1-(1-r_i)p)^{t-1}$$

where f_i denotes the probability that r takes on the value r_i, *i.e.* f is the probability distribution function (or the discrete approximation of the density function) for r. This probability distribution function can be used to derive a one-sided confidence interval for T. In other words, f can be used to derive the number of tests, U, required to bound the probability of failure by p with a given confidence C. The confidence interval is defined by:

$$\text{prob}(T \le U) = C$$

which yields, after some manipulation:

$$\sum_{i} f_i\ (1-(1-r_i)p)^{U} = 1-C$$

This formula is not as convenient to use as those derived earlier. In particular, it is not possible to solve for U directly, although U can be found with a variety of standard numerical methods.

The value of U in any particular set of circumstances is the quantity of interest. Many elements of the equation are unknown, and insight into the meaningful values of U can best be obtained by sensitivity analysis. The effects on U of changing the bound for p or the confidence level C are not immediately obvious. Similarly, the sensitivity of the analysis to changes in various characteristics of the r distribution is difficult to determine. In order to address such questions, we consider some possible distributions for r. In figure 5 we give two hypothetical distributions for r that are based on the following criteria:

(1) The limited empirical and anecdotal data indicate that the distribution for hidden failure probability has a ''bathtub'' shape in which most of the probability mass is near the value $r = 0$ and a small portion of the probability mass is near $r = 1$.

(2) Figure 4 indicates that it makes a substantial difference whether r is just near 1 or is exactly equal to 1. However, for low to moderate values of r the curve for $\frac{U}{\tilde{U}}$ is nearly flat, so these parts of the r distribution are not likely to have much effect on life testing reliability analysis. Thus the only difference between the hypothetical distributions is

---

prob (r=0) = 0.99	prob (r=0) = 0.99
prob (r=0.99) = 0	prob (r=0.99) = 0.01
prob (r=1) = 0.01	prob (r=1) = 0
a) r Distribution 1	b) r Distribution 2

Fig. 5. Hypothetical Distributions For r.

---

that distribution 1 has 1% of its mass at $r = 1$ and distribution 2 has 1% of its mass at $r = 0.99$.

Figure 6 shows how U and $\tilde{U}$ vary for a range of values of the bound for p. Both hypothetical distributions for p are shown and both parts of the figure were computed with a confidence level (C) of 0.989. Notice that U and $\tilde{U}$ vary uniformly with the bound for p. Recall that the hypothetical distributions for r differ from $r=0$ over only 1% of the r values. Nonetheless U and $\tilde{U}$ are quite distinct; there is a noticeable increase in the number of tests required to achieve a particular value of the bound for p.

The most striking aspect of figure 6 is that the graphs for the two hypothetical distributions are nearly identical. For the confidence level shown and for smaller confidence levels, the

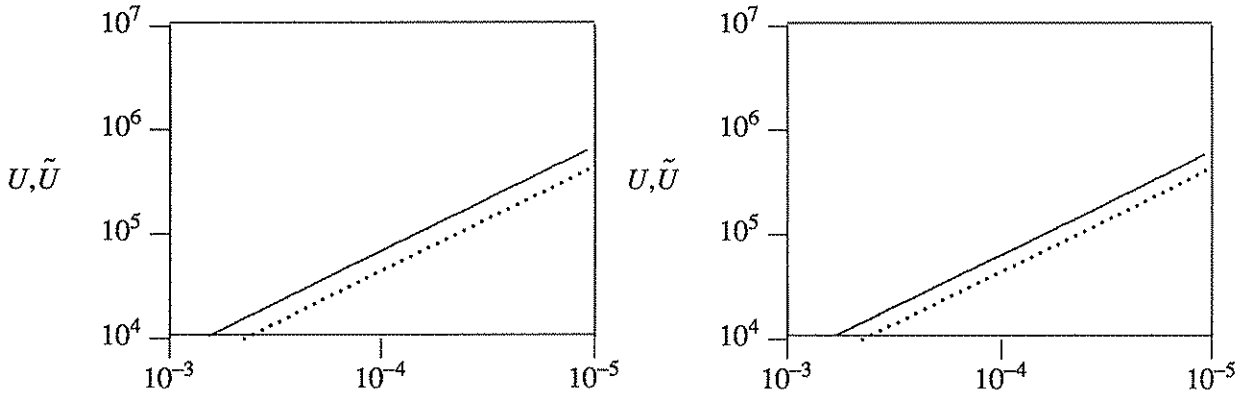
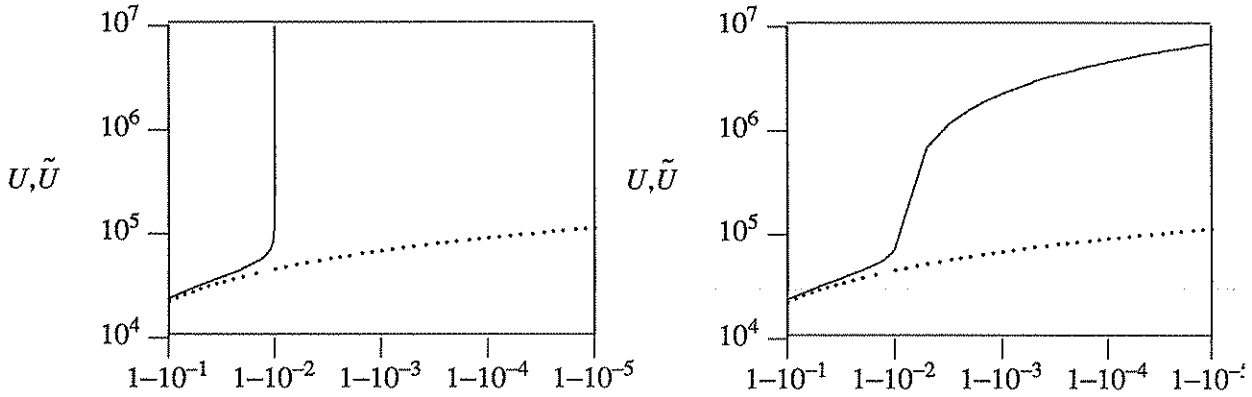


Fig. 6. U (solid), $\tilde{U}$ (dotted) vs. p For Two r Distributions (C=0.989).

number of tests needed to achieve a given bound on p is not seriously affected by whether the worst pseudo-oracles have r values near 1 or exactly equal to 1. However, the confidence level used for figure 6 was deliberately chosen to be less than 0.99, for reasons that are explained below.

The impact of altering the confidence level C, by contrast, is dramatically different for the two hypothetical distributions. Figure 7 compares the relationship between U and C with that between $\tilde{U}$ and C. The comparison is done for each of the hypothetical r distributions and with the bound $p = 10^{-4}$. Figure 7a shows $U,\tilde{U}$ vs. C for r distribution 1, in which the probability that $r=1$ is nonzero.



a) r Distribution 1

b) r Distribution 2

Fig. 7. U (solid), $\tilde{U}$ (dotted) vs. C For Two r Distributions ($p=10^{-4}$).

An important result of this analysis is that certain confidence intervals are unobtainable. For example, notice that it is not possible to achieve a confidence level above $C=0.99$ for the r distribution shown used in figure 7a.. The limit arises because, 1% of the time, $r=1$, so the failure domain of the program is completely obscured. For these 1% of possible systems, testing is useless and establishes no bound on the probability of failure. Thus no bound can be established with more than 99% confidence.

Figure 7b shows $U, \tilde{U}$ vs. C for r distribution 2, in which the probability that $r=1$ is zero. In this case, any desired confidence level is achievable. As the figure shows, however, two orders of magnitude more tests might be required to establish the bound with a specific confidence level using a pseudo-oracle.

The analysis of the two hypothetical distributions reveals that:

(1) The number of test cases needed to achieve a given bound on the probability of failure is affected by the probability of a large value of r. However, whether the value of r is equal to or just near 1 for this part of the r distribution is immaterial.

(2) The confidence level achievable for a particular r distribution is absolutely limited by $1 - \text{prob}(r=1)$. The distribution of r values near 1 dramatically affects the number of test cases needed to achieve confidence levels near $(1 - \text{prob}(r \approx 1))$ but any confidence level can be achieved if a sufficient number of tests are performed.

(3) The part of the r distribution for which $r \approx 1$ has a substantial effect on life testing reliability analysis.

5. CONCLUSION

We maintain that error detection must always be viewed as potentially imperfect. When life testing is performed to estimate the reliability of a software system, the effect of imperfect error detection is to make the number of observed failures (which would usually be zero) an inaccurate measure. Failures might have occurred that went unnoticed. Such undetected failures mean that bounds on reliability computed using the traditional life testing analysis can be incorrect to an arbitrarily large degree.

In this paper we have shown how to incorporate information about the quality of a pseudo-oracle into life testing analysis. The parameter that measures the fallibility of a pseudo-oracle is the hidden failure probability r. The analysis incorporating a specific value of r into the life testing model is illuminating theoretically, but not of immediate practical interest because the actual value of r for a given system is generally unknown.

Although the value of r for a particular program and pseudo-oracle is not available during testing, we think that it is reasonable for software developers to gather empirical data about r values using failure data for operational software systems. Our analysis shows that it is not necessary to know all the details of the distribution because life testing reliability estimation is sensitive only to the portion of the distribution at or near 1. Certainly a software reliability estimate based on an empirically estimated r distribution will be more realistic than one based on the assumption that r is always 0, as is implicit in traditional life testing analysis. At the very least, anyone who presents evidence supporting a particular reliability confidence interval should reveal and defend the assumed r distribution. The analysis presented in this paper can be used to make quantitative adjustments to reliability estimates if characteristics of the r distribution are broadly known.

The effect of incorporating uncertainty about the pseudo-oracle into the software assessment process is, of course, to increase the number of tests necessary to establish a given confidence interval. Somewhat surprisingly, our analysis shows that the number of tests necessary to support a given failure probability bound increases linearly as the desired bound increases, and that any bound can be established given a sufficient number of tests. However, our results show much stronger limits with respect to the confidence level that can be placed in the bound on failure probability. The portion of the r distribution near 1 may cause a dramatic increase in the number of tests necessary to achieve a given confidence bound. Finally, the achievable confidence level is absolutely limited by the portion of the distribution that is exactly equal to 1. Thus for perfectly reasonable r distributions, the confidence levels needed for critical software may be extremely difficult or impossible to achieve.

REFERENCES

[1] S.S. Brilliant, "Testing Software Using Multiple Versions", PhD Dissertation, University of Virginia, January, 1988.

[2] R.W. Butler, G.B. Finelli, "The Infeasibility of the Experimental Quantification of Life-Critical Software Reliability", *Proceedings ACM SIGSOFT '91: Software For Critical Systems*, 1991.

[3] Federal Aviation Administration, "System Design Analysis", Advisory Circular AC-25.1309-1, U.S. Department of Transportation, September 7, 1982.

[4] N.G. Leveson, P.R. Harvey, "Analyzing Software Safety", *IEEE Transactions On Software Engineering*, Vol SE-9, No. 5, September, 1983.

[5] R.J. Lipton, "New Directions in Testing", *Interface '90*, East Lansing, MI, May, 1990.

[6] D.R. Miller, "The Role of Statistical Modeling and Inference in Software Quality Assurance", *CSR Workshop on Software Certification*, Gatwick, England, September, 1988.

[7] D.R. Miller, "Making Statistical Inferences about Software Reliability", *NASA CR 4197* December, 1988.

[8] K.W. Miller, L.J. Morell, R.E. Noonan, S.K. Park, D.M. Nicol, B.W. Murrill, J.M Voas, "Estimating the Probability of Failure When Testing Reveals No Failures", *IEEE Transactions On Software Engineering*, Vol 18, No. 1, January, 1992.

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

[10] D.L. Parnas, J. van Schouwen, and S.P. Kwan, "Evaluation of Safety Critical Software", *Communications of the ACM*, Vol. 33, No. 6, June, 1990.

[11] D.P. Siewiorek, R.S. Swarz, *The Theory and Practice of Reliable System Design*, Bedford, MA: Digital Press, 1982.

[12] T.A. Thayer, M. Lipow, E.C. Nelson, *Software Reliability*, North Holland, 1978.

[13] E.J. Weyuker, "On Testing Non-testable Programs", *The Computer Journal*, Vol. 25, No. 4, November, 1982.