Estimating Software Reliability Using Inverse Sampling

Balwant Singh, Román Viveros and David L. Parnas

Abstract—**This paper addresses one of the perpetual questions faced by software developers, "How much testing is enough testing?". Software testing accounts for a substantial portion of software development costs, but releasing software with unacceptable reliability is also very costly. We begin with a discussion of how classical notions of reliability can be applied to deterministic software and explain that the reliability of a software is as much a function of the way that it is used as of quality of the software. We then illustrate how simple operational profiles can be used to characterize usage patterns. Finally, we show how the reliability requirements can be used to determine how much testing is necessary and whether or not the software is acceptable.**

Using the method of inverse sampling, also called negative binomial sampling, we develop an efficient statistical procedure for quantifying the reliability of a piece of software. The procedure allows substantial reductions in the average number of executions run over the traditional binomial testing. Other issues such as the calculation of upper confidence bounds for software failure rate under both binomial and negative binomial sampling are also addressed. The results obtained are illustrated numerically and graphically on several cases arising in practice. Some issues for further work, namely the use of sequential testing, the computer implementation of the methods developed and the testing of continuously-run software, are also discussed.

Index Terms—**Average number of executions, binomial sampling, negative binomial sampling, operational profile, software failure rate, software reliability, test case, test of hypotheses, trace, upper confidence bound.**

Balwant Singh is with the Communications Research Laboratory, McMaster University, Hamilton, Ontario, Canada L8S 4K1. e-mail: gill@mcserg.crl.mcmaster.ca.
Román Viveros is with the Department of Mathematics and Statistics, McMaster University, Hamilton, Ontario, Canada L8S 4K1. e-mail: rviveros@icarus.math.mcmaster.ca.
David L. Parnas is with the Communications Research Laboratory, McMaster University, Hamilton, Ontario, Canada L8S 4K1. e-mail: parnas@qusunt.crl.mcmaster.ca.

I. Introduction to Software Reliability

A. What is Software Reliability?

The word “reliability” is used in several different ways by the software development community. Some simply use it as a synonym for correctness. Another, very common, approach is to use “reliability” as a measure of the number of errors per thousand of lines. We prefer the classical interpretation of reliability, a measure of the probability that a product will work correctly when called upon.

It is frequently argued that the classical sense of “reliability” cannot apply to software because programs do not fail in the classical sense, i.e. they do not break or wear-out. Software failures are caused by faults that were present from the start. However, while most programs are incorrect, they fail only occasionally and it is useful to know the likelihood of a failure. In a world where perfect software remains a dream, it is useful to predict and compare product failure rates.

It is also argued that the classical meaning of reliability cannot apply to software because most programs behave deterministically and stochastic models do not hold. Software failure is not a random process; with enough effort we could (in theory) explain and predict which inputs to the program would lead to failure. However, we are unable to predict which inputs the user will supply to the software. User behavior can, and should, be modeled as a stochastic process. Software reliability is a measure of the probability that the input supplied to the software is one that will result in correct behavior. For software considered reliable, inputs leading to failure are very rare; conversely software is considered unreliable if failure-causing inputs occur frequently.

TABLE I
A Sample Portion of Code

```
IF (FLAG[1] = 0)
     THEN A1 .   .   .   ;
IF (FLAG[2] = 0)
     THEN A2 .   .   .   ;
IF (FLAG[3] = 0)
     THEN A3 .   .   .   ;
          ●
          ●
          ●
IF (FLAG[50] = 0)
     THEN A50 .   .   .   ;
```

It may be argued that a complete solution to the software reliability problem would be to test every admissible input and debug those resulting in failure. While this may be a feasible and often resorted to approach in small programs, for most applications the number of admissible inputs is prohibitively large. As an illustration, consider the portion of code depicted in Table I ([11]). The `FLAG` array itself will generate $2^{50} = 1.13 \times 10^{15}$ different inputs, their complete testing and debugging will be a challenge for most of the computers available today.

The foregoing makes it clear that we cannot talk about the reliability of a software product as if it were a property of the product alone. Instead, we must recognize that the reliability is a function of both the quality of the product and the way that the product is used. No software reliability estimate is meaningful unless it is associated with specified assumptions about how the product will be used. We will describe our assumptions by giving an "operational profile" which quantifies the likelihood of issuing each input.

The idea of an operational profile and its effect on software reliability estimation is easily illustrated with a simple example. Consider that we are using a stack with three possible operations, `PUSH(a)` puts the value of `a` on top of the stack, `POP` removes the value that is on top of the stack, and `TOP` returns to its caller the value that is on top of the stack. Suppose now that there is a bug in "top" so that two successive calls on that routine will get different answers, the second one being wrong. If we were to do uniform random testing on the assumption that at any point, all invocations are equally likely, we would find a low reliability for this module. However, in most situations it is rare to call "top" twice in a row because we know that "top" does not change what is on top of the stack so why call it again. Further, the probability of calling top after calling `PUSH(a)` is very low, because we know what we just put on top of the stack. If you were to test with a probability distribution that reflected these facts, the reliability of the module would look high simply because the situation that caused the error would rarely arise.

B. Software Products With Memory

Early programs were "stateless". That is, each time that they received an input they produced an answer that was not influenced by previous inputs. For stateless programs an input can be viewed as a single set of data read before the program runs. In stateless programs the requirements of a program can be expressed as a mathematical relation whose domain comprises the set of possible inputs and range comprises the set of acceptable outputs. This can be characterized by a predicate on (input, output) pairs.

Most modern software products are not stateless. They receive a sequence of inputs and produce a sequence of outputs. A failure may actually be "caused" by data that was received long before the failure can be actually detected by looking at the outputs.

For programs with internal states, it is necessary to specify and describe behavior in terms of traces. Each trace is a sequence consisting of alternating input values and output values. The acceptable behavior can be specified by giving a predicate on the set of traces.

A trace describes a software failure if the value of the predicate on the trace is false.

In this article we deal primarily with programs that have states and assume that we have been given a specification in terms of traces. In our analysis, we focus on programs that are used by initializing them, supplying them with an input sequence and then terminating them until needed, at which point we initialize again. In these situations, which are very common, the effect of errors can not be carried over to future executions. We discuss extensions of our method for other situations in section V.

C. Relevant Previous Work

C.1. Specifications in Terms of Traces

Work on specifying the required behavior of modules with internal state using assertions about traces began in 1977 with [1], but had its roots in earlier papers on "algebraic" specification such as [3]. Since that time many people have been interested in the problem, with [13, 17, 5, 12, 15, 4] yielding results that provide input to the present work. Wang's work [17] in particular lead to a simulator that can be used to determine whether or not an implementation satisfies a specification.

C.2. Reliability Estimation Using Traces

In several papers, (primarily [19]), Woit has discussed the problem of estimating reliability of a module after a long series of successful tests. Woit introduces the idea of using a set of conditional probabilities (an operational profile) to generate traces whose statistical characteristics are those that can be expected when the module is actually used. After testing the module with these traces, statistical methods can be used to estimate the reliability of the module if it is used in a way consistent with the operational profile. This paper proposes an improvement to Woit's binomial sampling approach by considering the inverse sampling of test cases.

C.3. MRET

Li ([9]) combines the work of [19] and [17] and describes a tool that can be used for reliability estimation. Li's Module Reliability Estimation Tool (MRET) generates test cases according to Woit's model, uses these to test the module, and evaluates the results using the simulator based on Wang's work. The purpose of this paper is to propose a way to improve the estimations obtained from Li's tool.

C.4. Test Case Generation

Interesting work on the generation of test cases has been carried out at the University of Tennessee at Knoxville. Using a Markovian assumption, under the direction of Professor Jesse Poore, [18, 16] have developed both theoretical frameworks and practical tools for generating traces.

II. User Operational Profile and Reliability

A. Operational Profile

We focus on software of a modular type, each module being composed of a set of possibly interacting individual programs for a specific area of application. Denote by M the module whose reliability we aim to quantify.

In order to make any reliability assessment relevant to a particular user of the module, the testing of the software has to take into account the patterns of usage specific to that user. The way the user accesses the module's programs is usually a non-deterministic process. Hence, resorting to probabilistic methods to describe the user's usage of the software becomes a natural choice. In general, the patterns of usage are captured in a quantitative manner by the operational profile distribution specific to the user.

Two basic ingredients compose the operational profile distribution:

- the set of all possible user module executions, also called test cases, to be denoted by $\boldsymbol{\tau}_M$; and
- a probability distribution over $\boldsymbol{\tau}_M$.

Following [19, 17, 9], $\boldsymbol{\tau}_M$ consists of all the sequences of events the user can possibly execute from the module, thus

$$\boldsymbol{\tau}_M = \{E_i, E_iE_j, E_iE_jE_k, ...\}.$$

The probabilities assigned to the executions in $\boldsymbol{\tau}_M$ should be indicative of the relative frequencies with which the user issues the executions in the normal course of software operation. For every user execution $I \in \boldsymbol{\tau}_M$, the corresponding probability will be denoted by $P(I)$. The two basic properties to make P[1] a proper probability distribution are:

(i) $0 \leq P(I) \leq 1 \quad \forall\ I \in \boldsymbol{\tau}_M$;

(ii) $\sum_{I \in \boldsymbol{\tau}_M} P(I) = 1$.

The pair $\{\boldsymbol{\tau}_M, P\}$ is called the *operational profile distribution.*

Although $\boldsymbol{\tau}_M$ may be relatively easy to describe for a given user, far more ingenuity is involved in the specification of the associated probability distribution. The following strategy can be useful for the latter.

(a) Suggest a probability distribution for the length L of the user executions.

[1]To remind the reader that P is a function, the notation $P(\cdot)$ is often used.

(b) Suggest a probability distribution over all the user executions of a given length.

Naturally, the probability distribution of L in (a) should reflect the relative frequencies of the lengths associated with the executions issued by the user in the normal operation of the module. In the absence of having such information available, one may resort to a probabilistic model. Because of its wide applicability in modeling natural phenomena, one could consider the truncated Poisson distribution, namely,

$$P(L = n) = \frac{e^{-\lambda}}{1 - e^{-\lambda}} \frac{\lambda^n}{n!}, \quad n = 1, 2, 3...,$$

where λ ($\lambda > 0$) is a parameter. Note that the average length of the executions issued by the user when the truncated Poisson distribution applies is $E(L) = \lambda/(1 - e^{-\lambda})$. One only needs to guess a reasonable value for parameter λ.

Regarding (b), using a well-known formula for conditional probability, the probability of issuing an execution of a given length n, $E_1E_2E_3...E_n$ say, can always be written as

$$P(E_1E_2E_3...E_n \mid n) = P(E_1)P(E_2 \mid E_1)P(E_3 \mid E_1E_2) \cdots P(E_n \mid E_1E_2...E_{n-1}).$$

Thus, given a prefix T such as $T = E_1E_2...E_{i-1}$, one needs to specify the conditional probabilities $P(E \mid T)$ for every possible event E to be issued by the user for execution immediately after T. Note that the actual probability with which the user issues execution $E_1E_2...E_n$ is

$$P(E_1E_2...E_n) = P(E_1E_2...E_n \mid n)P(L = n).$$

In most applications of practical interest, the number of prefixes is unwieldy. [19, 9] suggest and illustrate the tabulation of the conditional probabilities $P(E \mid T)$ using an equivalence class for the prefixes. A necessary condition for T and T^* to belong to the same class is that $P(E \mid T) = P(E \mid T^*)$ for every event E to follow T and T^*. Usually the classes of prefixes are characterized in terms of logical statements so that coding in automatic testing of the module is facilitated.

TABLE II
TRANSITION PROBABILITIES FOR THE STACK EXAMPLE

	Probability of next event being a:		
Last Event	`POP`	`PUSH`	`TOP`
`POP`	0.30	0.40	0.30
`PUSH`	0.40	0.57	0.03
`TOP`	0.60	0.39	0.01

A substantial simplification occurs when the selection of the events making up an input sequence is determined by the last event included in the sequence. This leads to a Markovian structure with the events as states and requiring only specification of the user initial selection probabilities $P(E_i)$ and the user transition probabilities $P(E_j \mid E_i)$. [18, 19, 16] use this approach to generate test cases randomly.

As an illustration, for the stack example of Section I.A, one could take $(0.01, 0.98, 0.01)$ as the user initial probabilities for `(POP,PUSH,TOP)` and user transition probabilities given by Table II[2].

B. Operational Failure Rate and Reliability

For a user of a module, particularly of a safety-critical one, it is of great interest to know how "reliable" the module is in actual operation. Although reliability may be defined and quantified in a variety of ways, we believe the following is the most relevant one from the perspective of the user.

Throughout this report, the *operational failure rate* p of a module M is defined as the probability that an execution of M issued by the user fails to run. This probability must be calculated from the operational profile distribution on the assumption that the user issues executions randomly from his/her operational profile. Defining $\omega(I)$ by

$$\omega(I) = \begin{cases} 0 \text{ if } I \text{ runs successfully,} \\ 1 \text{ if } I \text{ fails to run,} \end{cases}$$

for every $I \in \boldsymbol{\tau}_M$, then using basic properties of probability one obtains

$$p = E(\omega) = \sum_{I \in \boldsymbol{\tau}_M} \omega(I) P(I) = \sum_{I \in \boldsymbol{\tau}_M : \omega(I)=1} P(I). \tag{1}$$

The *operational reliability* q is defined as the probability that an execution issued by the user runs successfully. Naturally,

$$q = 1 - p.$$

For brevity, we will often use the terms failure rate and reliability in place of operational failure rate and operational reliability, respectively, in the sequel.

As an illustration, consider the stack example of Section I.A. Table III contains the exact failure rates, as calculated from (1), for two operational profile distributions. We assume that software failures take place only when two consecutive occurrences of `TOP` are encountered. Both operational profiles were set using a truncated Poisson distribution for the execution length and Markov transition probabilities, as discussed in Section II.A. The row labeled

[2]In this example, as in most real situations, the process is only approximately Markovian. The probabilities of `TOP` and `POP` are very low when the stack is empty, but this can not be represented in a simple transition matrix. To take this into account, the specification of the stack is needed to generate the test cases.

"Table II" uses $(0.01, 0.98, 0.01)$ as user initial probabilities for `(POP,PUSH,TOP)` and the transition probabilities of Table II. The row labeled "Uniform" uses a uniform distribution, that is, 1/3 for each initial and transition probability. Since under the uniform case the pair `(TOP)(TOP)` has a higher rate of occurrence, the same software is more prone to failure under the uniform operational profile. Also note that the higher λ is the higher the failure rate, this is so because when λ increases, the execution lengths tend to increase on an average, thus increasing the risks of including a `(TOP)(TOP)` pair somewhere in the execution.

TABLE III
Operational Failure Rate Illustrations for the Stack Example

λ	5	8	10	15	20
Table II	0.0033	0.0070	0.0096	0.0160	0.0224
Uniform	0.3130	0.4729	0.5590	0.7178	0.8195

[19] discussed (1). Note that in order to obtain the exact value of p, one needs to ascertain the value of $\omega(I)$ for every $I \in \boldsymbol{\tau}_M$. Naturally, $\omega(I)$ is revealed only after execution I is run and checked. Since in most realistic applications $\boldsymbol{\tau}_M$ is extremely large, the exact value of p, and hence of q, for a particular module M can never be calculated. The most we can hope for is to obtain a good estimate of p or to test statistical hypotheses about p. These tasks require the use of statistical methods.

Any statistical method employed to derive inferences about p will invariably require "success/failure" data obtained by running executions (test cases) randomly generated (sampled) using the operational profile distribution. The following realization is the starting point for any statistical analysis. If an execution I from $\boldsymbol{\tau}_M$ is generated at random from the operational profile distribution, then the outcome observed from the running of the execution is a Bernoulli trial with probabilities of "failure" and "success" equal to p and $q = 1 - p$, respectively.

In real applications, later stages of software development and testing will render decreasing values of p, with $p = 0$ being the ultimate gold value for any software developer. Commonly quoted user failure rate target values are $p = 10^{-2}$ to 10^{-3} for routine applications and $p = 10^{-4}$ to 10^{-9} for safety-critical modules. For instance, the Canadian and British nuclear energy regulating agencies often demand demonstration that the failure rate is $\leq 10^{-4}$ for the nuclear reactor shutdown software-based systems.

III. Binomial Testing

A. Binomial Sampling and Distribution

Assume that based on time, cost and possibly other considerations we are willing to run N test cases on the module. Using the operational profile distribution we generate N

executions randomly and with replacement from $\boldsymbol{\tau}_M$, $I_1, I_2, ..., I_N$ say, and run them to obtain the respective values $\omega(I_1), \omega(I_2), ..., \omega(I_N)$. Under these assumptions, the total number of failures observed among the N executions, X_N, namely

$$X_N = \sum_{i=1}^{N} \omega(I_i),$$

will have the *binomial distribution* with index N and probability parameter p. That is,

$$P(X_N = x) = \binom{N}{x} p^x (1-p)^{N-x}, \quad x = 0, 1, 2, ..., N, \tag{2}$$

where

$$\binom{N}{x} = \frac{N!}{x!(N-x)!}$$

is the familiar binomial coefficient. The mean and variance of X_N are $E(X_N) = Np$ and $Var(X_N) = Np(1-p)$, respectively. The inferential work about p of [19, 9] is based on (2).

B. Point and Interval Estimation of Failure Rate

Under the above mode of sampling, an estimator of p with good statistical properties is the sample proportion of failures $\widehat{p}_N$, that is

$$\widehat{p}_N = \frac{X_N}{N}. \tag{3}$$

Well-known results on binomial estimation (e.g., see [10, p. 256]) assure that $\widehat{p}_N$ is an unbiased estimator of p in the sense that $E(\widehat{p}_N) = p$ regardless of the value of p, with variance $Var(\widehat{p}_N) = p(1-p)/N$. The variance of $\widehat{p}_N$ is usually estimated by $\widehat{Var}(\widehat{p}_N) = \widehat{p}_N(1-\widehat{p}_N)/N$.

Although a failure rate estimator such as $\widehat{p}_N$ provides some empirical quantification of the true failure rate p, the estimator is subject to statistical variability. In the final analysis, the user's main concern is whether the module has an unacceptably high failure rate or not. This concern can be properly addressed by calculating an upper confidence bound for p or by testing statistical hypotheses about p.

Denote by x_{obs} the observed value of X_N. For given γ ($0 \leq \gamma \leq 1$, γ small), following [2, pp. 212–217], an upper confidence bound for p, denoted by p_U, at confidence level $1-\gamma$ is the largest value of p such that $P(X_N \leq x_{obs}) \geq \gamma$. It is shown in Appendix A that p_U is the unique root of the equation in p

$$\sum_{x=0}^{x_{obs}} \binom{N}{x} p^x (1-p)^{N-x} = \gamma. \tag{4}$$

In most cases, equation (4) requires a numerical solution.

In the later stages of development and testing of a module, typically a large number of executions N is run with no execution triggering a failure, i.e. $x_{obs} = 0$. In this case (4) becomes $(1-p)^N = \gamma$ yielding

$$p_U = 1 - \gamma^{1/N}.$$

For instance, if $N = 5,000$ random executions were run without failure, a 0.999 upper confidence bound ($\gamma = 0.001$) for p is $p_U = 0.0014$, i.e. with a level of confidence of 0.999 we can state that the module will exhibit at most 14 failures in $10,000$ executions run by the user.

C. Test of Hypotheses on Failure Rate

An alternative to calculating an upper confidence bound for the failure rate p is to test statistical hypotheses about p. Following [19, 9], the appropriate *null* and *alternative* hypotheses to consider, denoted by H_0 and H_1, respectively, are

$$H_0 : \ p \leq \theta \quad \text{vs.} \quad H_1 : \ p > \theta, \tag{5}$$

where θ is the largest value of the failure rate that the user is prepared to tolerate on the module. From the discussion in Section II.B, it follows that θ is usually a very small number.

The basis of the test will be the observed value of X_N, that is, the observed number of failures in N executions of the module generated at random from the operational profile distribution. The basic idea is to partition the range of X_N into two subsets, one denoted by C and called the *critical region* of the test, to be considered as the region of consonance or agreement with H_1. The other one is the complement of C in the range of X_N, to be denoted by C^*, considered as the region of consonance or agreement with H_0. Since for p small one would expect to observe a small count for X_N, then it is natural to take $C^* = \{0, 1, ..., x_0\}$ for some value x_0. Thus, $C = \{x_0 + 1, x_0 + 2, ..., N\}$. This choice of C is not only favored by intuition but is actually the "optimal" region in the sense of the Neyman-Pearson theory of testing statistical hypotheses (e.g., see [10, p. 299]). We still need to determine the value of x_0, the so called *critical point* of the test.

As a tester, our position is that if the observed count of failures falls in C then we take the view that the data provide statistical evidence in support of H_1 and against H_0, while when the observed count of failures falls in C^* then the statistical evidence is in support of H_0 and against H_1. Since the view adopted is based on the observed value of a random variable, it may be that H_0 is indeed true, i.e. $p \leq \theta$, but the observed count results in a value in C, thus suggesting that the data provide statistical evidence against H_0 when in fact H_0 is true. We call this misleading situation the *pessimistic position.* Naturally, the probability of occurrence of the pessimistic position, called the *false rejection risk* and denoted by $FRR_B(p; x_0, N)$, becomes a quantity of central interest in the statistical testing of (5). Note that

$$FRR_B(p; x_0, N) = P(X_N > x_0) = 1 - P(X_N \leq x_0) = 1 - \sum_{x=0}^{x_0} \binom{N}{x} p^x (1-p)^{N-x}, \quad p \leq \theta. \tag{6}$$

Similarly, when H_1 is true and the observed count of failures falls in C^* will lead to the misleading view that the data provide statistical evidence against H_1. We call this situation the *optimistic position*. The probability of occurrence of the optimistic position is called the *false acceptance risk*[3] and is denoted by $FAR_B(p; x_0, N)$. Thus

$$FAR_B(p; x_0, N) = P(X_N \leq x_0) = \sum_{x=0}^{x_0} \binom{N}{x} p^x (1-p)^{N-x}, \quad p > \theta. \tag{7}$$

It is shown in Appendix A that $FRR_B(p; x_0, N)$ is an increasing function of p while $FAR_B(p; x_0, N)$ is a decreasing function of p. Thus, their respective largest values are assumed when $p = \theta$, that is, they are $FRR_B(\theta; x_0, N)$ and $FAR_B(\theta; x_0, N)$. Note that $FRR_B(\theta; x_0, N) = 1 - FAR_B(\theta; x_0, N)$.

See last page.

Fig. 1. Plots of $FRR(p) = FRR_B(p; x_0, N)$ and $FAR(p) = FAR_B(p; x_0, N)$ for the particular case in which $\theta = 10^{-3}$, $x_0 = 0$ and $N = 5,000$.

Occurrence of the pessimistic position results in unnecessary additional review and testing of the module by the software developer. Its frequent occurrence will unduly tax the developer's time and resources. On the other hand, occurrence of the optimistic position will present risks for the user. These risks may translate in considerable loss, particularly in safety-critical applications. Ideally, one would like to reduce the possibility of making either error. However, because $FRR_B(\theta; x_0, N) = 1 - FAR_B(\theta; x_0, N)$, by choosing x_0 appropriately one can make one of them small at the expense of increasing the other one. It is possible, however, to make one of them small, $FAR_B(\theta; x_0, N)$ say, and keep $FRR_B(p; x_0, N)$ reasonably small at some chosen value $p = p_0 < \theta$.

As an illustration, consider the testing of (5) where the target failure rate is $\theta = 10^{-3}$. If resources permit running $N = 5,000$ random executions, then taking $x_0 = 0$ will result in $FAR_B(\theta; x_0, N) = 0.007$. Plots of $FRR_B(p; x_0, N)$ and $FAR_B(p; x_0, N)$ are displayed in Fig. 1.

Known as the *operating characteristic curve* or *power function*, the quantity

$$OC_B(p; x_0, N) = 1 - FAR_B(p; x_0, N), \quad p > \theta,$$

is often used in place of $FAR_B(p; x_0, N)$. Naturally, minimizing the false acceptance risk is equivalent to maximizing the associated operating characteristic curve.

[3]The pessimistic and optimistic positions are usually called *Type I* and *Type II* errors, respectively, in the statistics literature. Also, the probabilities of their occurrence are usually referred to as the *sizes* of the errors.

IV. Negative Binomial Testing

A. Negative Binomial Sampling and Distribution

Also known as inverse sampling, negative binomial sampling occurs when the experimenter continues sampling and testing until a pre-specified number of failures, r say, is completed. The random variable of interest here is the total number of test cases, denoted by Y_r, run until the r failures are completed. Naturally, when the failure rate p is large, Y_r will tend to be small, while for p small, Y_r will tend to take on large values.

Often called a "waiting time" random variable, Y_r has the *negative binomial* distribution with parameters r and p,

$$P(Y_r = y) = \binom{y-1}{r-1} p^r (1-p)^{y-r}, \quad y = r, r+1, \ldots \tag{8}$$

The expected number of executions required is $E(Y_r) = r/p$ and the variability in the number of executions required is $Var(Y_r) = r(1-p)/p^2$. For details, see [7, pp. 120–122].

Perhaps the most striking difference between binomial and negative binomial modes of sampling is the fact that the total number of executions run under the latter is unknown prior to conducting the testing. This may be considered a drawback since the resources available usually dictate the amount of testing possible. We will show, however, that when the aim is to test statistical hypotheses about the failure rate p such as (5), one can bound the number of executions to be run under negative binomial sampling prior to conducting the software testing. Moreover, under this provision, the negative binomial method exhibits superior performance as measured by the average number of module executions run when testing the hypotheses. See Section IV.D for details.

A remarkably useful relationship between the negative binomial and the binomial distributions stems from the obvious fact that the rth software failure will occur after the yth execution when and only when less than r failures occurred in the first y executions run. In mathematical terms, this is captured by the equation

$$P(Y_r > y) = P(X_y < r), \tag{9}$$

where X_y is the binomial random variable that counts the total number of failures observed in the first y executions run. This probabilistic relationship holds for any r and y with $y \geq r$. Equation (9) is often called the *fundamental identity.*

B. Point and Interval Estimation of failure Rate

Under negative binomial sampling, a commonly used estimator of the failure rate p is

$$\widehat{p}_{NB} = \frac{r-1}{Y_r - 1}. \tag{10}$$

[8, pp. 593-594] show that $\widehat{p}_{NB}$ is an unbiased estimator of p with variance

$$Var(\widehat{p}_{NB}) = \frac{p^2(1-p)}{r}\left[1 + \frac{2(1-p)}{r+1} + \frac{6(1-p)^2}{(r+1)(r+2)} + ...\right] \tag{11}$$

An estimate of this variance is obtained by replacing p by $\widehat{p}_{NB}$.

Denote by y_{obs} the observed value of Y_r. For given γ ($0 \leq \gamma \leq 1$, γ small), again applying [2, pp. 212–217], an upper confidence bound, p_U, for the failure rate p at confidence level $1-\gamma$ is the largest value of p such that $P(Y_r \geq y_{obs}) \geq \gamma$. The results from Appendix A assure that p_U is the unique solution for p in the equation

$$\sum_{y=r}^{y_{obs}-1} \binom{y-1}{r-1} p^r (1-p)^{y-r} = 1-\gamma. \tag{12}$$

The fundamental identity (9) implies that $P(Y_r \geq y_{obs}) = P(X_{y_{obs}-1} \leq r-1)$ so that p_U can alternatively be determined by solving for p in the equation

$$\sum_{x=0}^{r-1} \binom{y_{obs}-1}{x} p^i (1-p)^{y_{obs}-i-1} = \gamma. \tag{13}$$

In situations where the software is expected to have a low failure rate, for instance in later stages of development and testing, one usually considers small values for r. Equation (13) is particularly useful in these situations. For instance, if $r=1$, equation (13) becomes $(1-p)^{y_{obs}-1} = \gamma$ yielding

$$p_U = 1 - \gamma^{1/(y_{obs}-1)}.$$

As an illustration, if the first failure observed ($r=1$) occurred at the $8,000$th execution selected at random from the operational profile distribution, an upper confidence bound for the failure rate at confidence level 0.999 ($\gamma = 0.001$) is $p_U = 0.0009$, i.e. with confidence 0.999 we can state that a software failure will occur in at most 9 cases out of $10,000$ executions run by the user.

C. Test of Hypotheses on failure Rate

Consider again the testing of hypotheses (5) now assuming a negative binomial mode of sampling. The basis of the test is Y_r with r being specified in advance. Applying the Neyman-Pearson theory of testing statistical hypotheses (e.g., see [10, p. 299]), the optimal critical region is $C = \{r, r+1, ..., y_0\}$ so that $C^* = \{y_0+1, y_0+2, ...\}$, for appropriate choice of y_0. For this C, the probability of falling in the pessimistic position, $FRR_{NB}(p; y_0, r)$, is

$$FRR_{NB}(p; y_0, r) = P(Y_r \leq y_0) = \sum_{y=r}^{y_0} \binom{y-1}{r-1} p^r (1-p)^{y-r}, \quad p \leq \theta. \tag{14}$$

Similarly, the probability with which one falls in the optimistic position is

$$FAR_{NB}(p; y_0, r) = P(Y_r \geq y_0 + 1) = 1 - \sum_{y=r}^{y_0} \binom{y-1}{r-1} p^r (1-p)^{y-r}, \quad p > \theta. \tag{15}$$

Using the fundamental identity (9) gives $P(Y_r \leq y_0) = 1 - P(Y_r > y_0) = 1 - P(X_{y_0} < r)$. Thus

$$FRR_{NB}(p; y_0, r) = 1 - \sum_{x=0}^{r-1} \binom{y_0}{x} p^x (1-p)^{y_0-x}, \quad p \leq \theta. \tag{16}$$

Similarly, $P(Y_r \geq y_0 + 1) = P(Y_r > y_0) = P(X_{y_0} < r)$, yielding

$$FAR_{NB}(p; y_0, r) = \sum_{x=0}^{r-1} \binom{y_0}{x} p^x (1-p)^{y_0-x}, \quad p > \theta. \tag{17}$$

Equations (16) and (17) are not only numerically convenient, particularly when r is small, but more importantly, they provide the means for comparing the performance of the binomial and the negative binomial modes of sampling on the testing of (5). See Section IV.D for details. As an illustration of the convenience of (16) and (17), for $r = 1$ they give

$$FRR_{NB}(p; y_0, r = 1) = 1 - (1-p)^{y_0}, \quad p \leq \theta; \quad FAR_{NB}(p; y_0, r = 1) = (1-p)^{y_0}, \quad p > \theta.$$

Another application of (16) and (17) is to show that $FRR_{NB}(p; y_0, r)$ is an increasing function of p and that $FAR_{NB}(p; y_0, r)$ is a decreasing function of p. See Appendix A for details. As a result, the largest incidences of the pessimistic and optimistic positions occur when $p = \theta$.

D. Average Number of Executions

We mentioned earlier the possibility of having to run a prohibitively large number of executions in order to get the value of Y_r. While this is true in general, when testing statistical hypotheses about p we only need to ascertain whether Y_r falls in C or in C^*. If the observed value of Y_r falls in C, at most y_0 executions would have been run while if by the y_0th execution we have not completed the r failures then unequivocally we know that Y_r will fall in C^* necessitating no more executions. In summary, in order to reach a decision regarding the outcome of the test, at most y_0 executions will be required.

Denote by M_r the total number of executions needed. It follows from the above that M_r is a random variable with distribution

$$\begin{aligned} P(M_r = m) &= P(Y_r = m) = \binom{m-1}{r-1} p^r (1-p)^{m-r}, \quad r \leq m \leq y_0 - 1; \\ &= P(Y_r \geq y_0) = 1 - \textstyle\sum_{y=r}^{y_0-1} \binom{y-1}{r-1} p^r (1-p)^{y-r}, \quad m = y_0. \end{aligned}$$

Thus, the *average number of executions* required, $ANE_{NB}(p; y_0, r)$, is

$$ANE_{NB}(p; y_0, r) = E(M_r) =$$

$$\sum_{m=r}^{y_0} m \binom{m-1}{r-1} p^r (1-p)^{m-r} + y_0 \left[1 - \sum_{m=r}^{y_0} \binom{m-1}{r-1} p^r (1-p)^{m-r}\right], \quad 0 \le p \le 1. \tag{18}$$

One can readily verify that $ANE_{NB}(p; y_0, r)$ is a decreasing function of p. See Appendix B for details. Naturally, for the binomial model, $ANE_B(p; x_0, N) = N,\ 0 \le p \le 1$.

A computationally convenient alternative expression for (18), particularly useful when r is small, is the formula

$$ANE_{NB}(p; y_0, r) = \frac{r}{p}\left[1 - \binom{y_0}{r} p^r (1-p)^{y_0-r+1}\right] + \left(y_0 - \frac{r}{p}\right) \sum_{i=0}^{r-1} \binom{y_0}{i} p^i (1-p)^{y_0-i}. \tag{19}$$

See Appendix B for details. For instance, applying (19) to the case $r = 1$ yields

$$ANE_{NB}(p; y_0, 1) = \frac{1-(1-p)^{y_0}}{p}.$$

As an illustration, consider the testing of (5) with the target failure rate being $\theta = 10^{-3}$. If we set $r = 1$, then $y_0 = 7,598$ will ensure that the largest incidence of the optimistic position will be $FAR_{NB}(10^{-3}; 7598, 1) = 0.0005$ or a maximum of 5 times out of 10,000 instances in which $p > 10^{-3}$. Fig. 2 displays the average number of executions for this situation. Note that $ANE_{NB}(p; 7598, 1)$ decreases rapidly with p increasing. At the target failure rate, we average $ANE_{NB}(10^{-3}; 7598, 1) = 1,000$ executions.

See last page.

Fig. 2. Plot of $ANE(p) = ANE_{NB}(p; y_0, r)$ for the particular case in which $\theta = 10^{-3}$, $y_0 = 7,598$ and $r = 1$.

E. Binomial Vs. Negative Binomial

In planning a software testing project, the software engineer needs to decide what mode of sampling renders the best results given the constraints in time and resources imposed on the project. In most cases, after careful consideration of all the elements at hand, the engineer will produce a bound on the number of executions affordable under the circumstances[4]. Let B denote this bound.

[4]Note that this limit is often a "soft" one. If the Engineer can show that the limit is insufficient to provide adequate information, it may be possible to get money for additional testing. The methods discussed in this paper can provide that evidence.

TABLE IV

Average Number of Executions Run Under the Negative Binomial Mode of Sampling for Several Target Failure Rates θ and Values of r of Practical Interest With $\beta_0 = 0.001$

θ	$r = x_0 + 1$	$y_0 = N = B$	$ANE_{NB}(\theta; y_0, r)$
10^{-4}	1	69,074	9,990
10^{-3}	1	6,904	999
10^{-2}	1	687	100
10^{-1}	1	66	10
10^{-4}	2	92,330	19,987
10^{-3}	2	9,228	1,999
10^{-2}	2	919	200
10^{-1}	2	88	20
10^{-4}	3	112,284	29,988
10^{-3}	3	11,224	2,999
10^{-2}	3	1,118	300
10^{-1}	3	107	30
10^{-4}	4	130,617	39,987
10^{-3}	4	13,057	3,999
10^{-2}	4	1,301	400
10^{-1}	4	125	40

When two or more statistical procedures for testing (5) are available, all having the same false acceptance risk at the target failure rate $p = \theta$, the software engineer would like to decide which one is more economical in the long run. In other words, the engineer would like to find out which statistical procedure requires the smallest average number of executions to reach a decision regarding the plausibility of H_0 and H_1. We address this issue in this section for the binomial and negative binomial modes of sampling.

Let β_0 denote the false acceptance risk that we aim to achieve at the target failure rate $p = \theta$. Consider the binomial mode of sampling with $N = B$. Assume x_0 is such that $FAR_B(\theta; x_0, N) = \beta_0$. The false rejection risk for $p \leq \theta$ and the false acceptance risk for $p > \theta$ are given by (6) and (7), respectively. Taking $r = x_0 + 1$ and $y_0 = N$ in the negative binomial mode of sampling, using (16) and (17) one can readily verify that $FRR_{NB}(p; y_0, r) = FRR_B(p; x_0, N)$ for all $p \leq \theta$ and $FAR_{NB}(p; y_0, r) = FAR_B(p; x_0, N)$ for all $p > \theta$. In other words, both modes of sampling produce identical results as far as the false rejection and false acceptance risks are concerned. In particular, $FAR_{NB}(\theta; y_0, r) = \beta_0$. By contrast, consider the number of executions involved for the above choices of N, x_0, r and y_0. In the case of the binomial mode of sampling, this number is fixed and equals $N = B$. For the negative binomial mode of sampling, the average number of executions required, $ANE_{NB}(p; y_0, r)$, is given by (18) or (19). This number never exceeds $y_0 = N$ and in the cases of practical interest, as illustrated by Fig. 2, decreases rapidly as p increases. Other cases of practical relevance are considered in Table IV. Note that the average number of executions run at the target failure rate $p = \theta$ under the negative binomial mode of sampling is never more than 32% of the number of executions run under the binomial mode of sampling in all the cases considered in Table IV.

See last page.

Fig. 3. Plots of $ANE(p) = ANE_{NB}(p; y_0, r)$ for the particular cases considered in Table IV in which $\theta = 10^{-3}$.

Fig. 3 presents a wider view of the negative binomial performance on the four cases considered in Table IV for a target failure rate of $\theta = 10^{-3}$. The salient overall point of the comparison is that, unless the software is nearly perfect, the negative binomial mode of sampling brings about large reductions in the average number of executions over the binomial mode of sampling for identical false rejection and false acceptance risks.

V. Future Work

A. Sequential Testing

Placing a tight control on the false acceptance risk $FAR(p)$ when testing (5) provides protection to the software user against the risk of receiving an unacceptable product from the developer. For instance, if $FAR(p) \leq 0.001$ for all $p > \theta$, the false acceptance risk is at most 1 in $1,000$ on an average of receiving a bad product, i.e. a module with a failure rate larger than the target value θ. Because of the catastrophic consequences of failures, this protection is essential in safety-critical software.

Both binomial and negative binomial testing accommodate the above requirement. However, a price is paid in the form of an extremely large false rejection risk. This can be seen from the fact that $FRR(p)$ approaches $FRR(\theta) = 1 - FAR(\theta)$ as $p \to \theta$ for both modes of testing. For instance, when $FAR(\theta) = 0.001$, $FRR(p)$ takes on values close to 0.999 for p near θ. It should be stated here that this undesirable feature is characteristic of practically every procedure for testing (5) based on the Neyman-Pearson theory (e.g., see [14, pp. 129–130]).

Another undesirable characteristic of most of the standard procedures for testing (5) is their requirement of a fixed number of software executions for their validation. This applies to binomial testing but to a much lesser extent to negative binomial testing.

Ideally one would like to have a statistical procedure that:

(a) controls $FRR(p)$ and $FAR(p)$ satisfactorily; and

(b) uses efficiently the statistical information gained from every execution run and calls for a halt to testing as soon as sufficient evidence to support H_0 or H_1 is amassed.

The sequential probability ratio procedure is designed primarily to comply with (b) and at the same time to have a good handle on (a). For details, see [14, Ch. 8]. We are currently investigating this procedure to test hypotheses (5).

B. Automatic Implementation

In order to make efficient use of the methods presented in this report, one needs to implement them in a computer program that automatically performs the various components of software testing. These include generation of test cases, test harness, running of test cases, checking of results and reliability assessment. In [9] Li developed a prototype black-box automated testing tool, called Module Reliability Estimation Tool (MRET), that performs these tasks. Li's tool combines the work of the Software Engineering Research Group at McMaster University on the trace assertion method and the reliability assessment results on binomial testing of (5). The latter results are precisely those described in Section III.C.

MRET's flexible structure will permit a smooth implementation of the procedures developed in this report. As part of the binomial analysis already in MRET, one could add the calculation of the upper confidence bound for the failure rate at a desired level of confidence based on the observed data (Section III.C). Plots of the false rejection and false acceptance risks, i.e. $FRR_B(p; x_0, N)$ and $FAR_B(p; x_0 N)$, would also be informative (Section III.C).

An alternative choice in MRET's reliability assessment menu would be to conduct a negative binomial analysis instead. The basic output results here would be an upper confidence bound for the failure rate (Section IV.B) and the results of the testing of (5) (Section IV.C), including plots of $FRR_{NB}(p; y_0, r)$ and $FAR_{NB}(p; y_0, r)$. The count $y_0 - M_r$ will indicate the savings in terms of the number of executions run over the binomial testing for the same false rejection and false acceptance risks.

We feel that adding to MRET the statistical procedures presented in this report will enhance considerably MRET's reliability assessment power.

C. On Execution Length and Software Reliability Measures

Different users will exhibit diverse length (L) patterns for the executions they put to the software, with the nature of the application accounting for a large share in characterizing the shape of these patterns. Software design and programming methods are also important factors. In view of the unpredictability of L for most users, we modeled L as a random variable. Flexibility in selecting a probabilistic model for L will permit accommodation of a variety of execution length patterns.

Two extreme situations arise in analyzing the execution length patterns. First, consider a user whose demands are always of the same length, n_0 say. Our approach handles this situation by defining an atomic distribution for L concentrated at n_0, that is $P(L = n_0) = 1$ and $P(L = n) = 0$ for $n \neq n_0$.

Second, consider an application where the software is run continuously, thus one may assume that $L = \infty$ with probability 1. A substantial number of important applications falls in this category, including control systems for nuclear plants, chemical plants, air traffic and bank transactions, among others. Reinitializations of the system in these applications take place only when a software failure occurs or when software upgrades of major components are installed. Since only a small number of fairly long executions are possible in this case, the notion of failure rate based on repeated executions is not applicable anymore. We are working on extending our methods to handle these situations by focusing on the traces for the individual events making up the executions.

Although in this article we considered the overall failure rate p of (1) to be the quantity of main interest, other measures of reliability may be of interest. One such quantity is the conditional failure rate among executions of a given length, n say. In the notation of Section II this quantity is given by

$$p_n = \sum_{I \in \tau_{M(n)}} \omega(I) P(I) / \sum_{I \in \tau_{M(n)}} P(I),$$

where $\tau_{M(n)}$ denotes the class of executions in τ_M whose length is n. The methods developed in this paper equally apply to p_n is we redefine the operational profile distribution in an obvious way so that only executions of length $L = n$ are sampled following the user profile.

Another measure of reliability is the *length reliability function*, $p_{LR}(n)$, defined as the probability that the software survives running the first n events without a failure given that the execution has length at least n. We can represent $p_{LR}(n)$ as

$$p_{LR}(n) = \sum_{I \in \tau_{M(\geq n)}} \omega^*(I) P(I) / \sum_{I \in \tau_{M(\geq n)}} P(I),$$

where $\tau_{M(\geq n)}$ is the class of executions in τ_M whose length is at least n, and $\omega^*(I) = 0$ for any $I \in \tau_{M(\geq n)}$ whose first n events are run without a failure and $\omega^*(I) = 1$ otherwise. Our methods of sampling and statistical analysis also apply to $p_{LR}(n)$ if we work with ω^* in place of ω and condition P to $\tau_{M(\geq n)}$.

Appendix A: Upper Confidence Bound Calculation and Monotonicity of Error Sizes

The key result used in this appendix is the following well-known expression for binomial probabilities given, for instance, in [6, formula (1.94)],

$$P(X_n \geq k) = k \binom{n}{k} \int_0^p t^{k-1}(1-t)^{n-k} dt, \qquad (20)$$

valid for any $k = 1, 2, ..., n$ and $0 \le p \le 1$. The right-hand-side of (20) can be recognized as the cumulative distribution function of the beta distribution with parameters $a = k$ and $b = n - k + 1$. In particular, the values it takes at $p = 0$ and 1 are 0 and 1, respectively.

Consider the determination of the upper confidence bound for p discussed in Section III.B. Using (20) gives

$$g_1(p) = P(X_N \le x_{obs}) = 1 - (x_{obs} + 1) \binom{N}{x_{obs} + 1} \int_0^p t^{x_{obs}} (1-t)^{N - x_{obs} - 1} dt, \quad 0 \le p \le 1,$$

yielding

$$g_1'(p) = -(x_{obs} + 1) \binom{N}{x_{obs} + 1} p^{x_{obs}} (1-p)^{N - x_{obs} - 1}, \quad 0 < p < 1.$$

Thus, $g_1'(p) < 0$ for all $0 < p < 1$ implying that $g_1(p)$ is a strictly monotonic decreasing function of p. Since $g_1(p)$ is continuous, $g_1(0) = 0$ and $g_1(1) = 1$, it follows that for every $0 \le \gamma \le 1$, the largest value of p satisfying $g_1(p) \ge \gamma$ always exists and is the unique solution to the equation $g_1(p) = \gamma$, i.e.

$$\sum_{x=0}^{x_{obs}} \binom{N}{x} p^x (1-p)^{N-x} = \gamma.$$

Applying a similar reasoning to $FRR_N(p; x_0, N) = P(X_N > x_0)$ of (6) yields

$$FRR_N'(p; x_0, N) = (x_0 + 1) \binom{N}{x_0 + 1} p^{x_0} (1-p)^{N - x_0 - 1}, \quad 0 < p < \theta.$$

Thus $FRR_N'(p; x_0, N) > 0$ for all $0 < p < \theta$, implying that $FRR_N(p; x_0, N)$ is strictly monotonic increasing in p. Similarly, the strictly decreasing monotonicity of $FAR_N(p; x_0, N)$ becomes apparent from the fact that

$$FAR_N'(p; x_0, N) = -(x_0 + 1) \binom{N}{x_0 + 1} p^{x_0} (1-p)^{N - x_0 - 1}, \quad \theta < p < 1.$$

Using the fundamental identity (9) one obtains $g_2(p) = P(Y_r \ge y_{obs}) = P(X_{y_{obs}-1} \le r - 1)$. Applying to $g_2(p)$ the reasoning we applied to $g_1(p)$ reveals that the upper confidence bound for p under negative binomial sampling is the unique solution for p in (12).

Appendix B: Decreasing Monotonicity of Average Number of Executions

Consider the average number of executions required to test hypotheses (5) under a negative binomial mode of sampling. The first term in (18) can be written as

$$\sum_{m=r}^{y_0} m \binom{m-1}{r-1} p^r (1-p)^{m-r} = \frac{r}{p} \sum_{m=r}^{y_0} \binom{m}{r} p^{r+1} (1-p)^{m-r} =$$

$$\frac{r}{p}P(Y_{r+1} \leq y_0) + r\binom{y_0}{r}p^r(1-p)^{y_0-r}.$$

Using the fundamental identity (9) we obtain

$$P(Y_{r+1} \leq y_0) = 1 - P(Y_{r+1} > y_0) = 1 - P(X_{y_0} < r+1),$$

yielding

$$\sum_{m=r}^{y_0} m\binom{m-1}{r-1}p^r(1-p)^{m-r} = \frac{r}{p}\left[1 - \sum_{i=0}^{r}\binom{y_0}{i}p^i(1-p)^{y_0-i}\right] + r\binom{y_0}{r}p^r(1-p)^{y_0-r}$$

The term in square-brackets in (18) is precisely $P(Y_r > y_0)$. Using again the fundamental identity (9) yields

$$1 - \sum_{m=r}^{y_0}\binom{m-1}{r-1}p^r(1-p)^{m-r} = \sum_{i=0}^{r-1}\binom{y_0}{i}p^i(1-p)^{y_0-i}$$

Replacing the above expressions in (18) leads to (19) after straightforward simplifications.

The summation term in (19) is clearly $P(X_{y_0} \leq r-1) = 1 - P(X_{y_0} \geq r)$. Using identity (20) leads to

$$\sum_{i=0}^{r-1}\binom{y_0}{i}p^i(1-p)^{y_0-i} = 1 - r\binom{y_0}{r}\int_0^p t^{r-1}(1-t)^{y_0-r}dt.$$

Replacing this in (19) and applying obvious simplifications yields

$$ANE_{NB}(p; y_0, r) = y_0 - r\binom{y_0}{r}p^{r-1}(1-p)^{y_0-r+1} - \left(y_0 - \frac{r}{p}\right)r\binom{y_0}{r}\int_0^p t^{r-1}(1-t)^{y_0-r}dt.$$

Taking the derivative with respect to p and applying straightforward simplifications in conjunction with (20) one obtains

$$ANE'_{NB}(p; y_0, r) = -\frac{r}{p^2}P(X_{y_0} \geq r+1), \quad 0 < p < 1.$$

Thus $ANE'_{NB}(p; y_0, r)$ is always negative unless $y_0 = r$, a case of no practical relevance. This shows that $ANE_{NB}(p; y_0, r)$ is strictly monotonic decreasing in p.

Acknowledgements

This research was supported in part by Bell Canada, the Telecommunications Research Institute of Ontario (TRIO) and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] W. Bartussek and D.L. Parnas, "Using Assertions About Traces to Write Abstract Specifications for Software Modules," UNC Report TR77-012, 1977, 26 pages.

[2] D.R. Cox and D.V. Hinkley, *Theoretical Statistics,* London: Chapman and Hall, 1974.

[3] J.V. Guttag and J.J. Horning, "The Algebraic Specification of Abstract Data Types," *Acta Informatica* **10**, pp. 27-52, 1978.

[4] D. Hoffman, "The Specifications of Communication Protocols,". *IEEE Transactions on Computers* **C-34**, No. 12, pp. 1102-1113, 1985.

[5] R. Janicki, "Foundations of the Trace Assertion Method of Module Interface Specification," CRL Report 348, NSERC, McMaster University, 1997.

[6] N.L. Johnson, S. Kotz and A.W. Kemp, *Univariate Discrete Distributions, Second Edition*, New York: Wiley, 1992.

[7] J.G. Kalbfleisch, *Probability and Statistical Inference, Vol. 1: Probability,* New York: Springer Verlag, 1985.

[8] M.G. Kendall and A. Stuart, *The Advanced Theory of Statistics, Vol. 2*, New York: Hafner Publishing Company, 1961.

[9] Ch. Li, *Documentation Based Testing Tool for Software Module Reliability Estimation,* Master of Engineering Thesis, McMaster University, 1986.

[10] B.W. Lindgren, *Statistical Theory,* New York: MacMillan, 1976.

[11] J.C. Munson and R.H. Ravenel, "Designing Reliable Software," in *Proceedings of the 4th International Symposium on Software Reliability Engineering,* 3–6 November 1993, Denver, Colorado, pp. 45-54.

[12] T.S. Norvell, "On Trace Specification," CRL Report 305, NSERC, McMaster University, 1995.

[13] D.L. Parnas and Y. Wang, "The Trace Assertion Method of Module Interface Specification," TR 89-261, Telecommunications Research Institute of Ontario (TRIO), Queen's University, 1989.

[14] S.D. Silvey, *Statistical Inference*, London: Chapman & Hall, 1975.

[15] K. Stencel, "Refine Simulation Techniques for the Trace Assertion Method," CRL Report 314, Telecommunications Research Institute of Ontario (TRIO), McMaster University, 1995.

[16] G.H. Walton, *Generating Transition Probabilities for Markov Chain Usage Models,* Ph.D. Thesis, University of Tennessee, 1995.

[17] Y. Wang, "Specifying and Simulating the Externally Observable Behavior of Modules," CRL Report 292, Telecommunications Research Institute of Ontario (TRIO), McMaster University, 1994.

[18] J.A. Whittaker, *Markov Chain Techniques for Software Testing and Reliability Analysis,* Ph.D Thesis, University of Tennessee, 1992.

[19] D.M. Woit "Operational Profile Specification, Test Cases Generation, and Reliability Estimation for Modules," CRL Report 281, Telecommunications Research Institute of Ontario (TRIO), McMaster University, 1994.