A User-Oriented Software Reliability Model

ROGER C. CHEUNG, MEMBER, IEEE

Abstract—A user-oriented reliability model has been developed to measure the reliability of service that a system provides to a user community. It has been observed that in many systems, especially software systems, reliable service can be provided to a user when it is known that errors exist, provided that the service requested does not utilize the defective parts. The reliability of service, therefore, depends both on the reliability of the components and the probabilistic distribution of the utilization of the components to provide the service. In this paper, a user-oriented software reliability figure of merit is defined to measure the reliability of a software system with respect to a user environment. The effects of the user profile, which summarizes the characteristics of the users of a system, on system reliability are discussed. A simple Markov model is formulated to determine the reliability of a software system based on the reliability of each individual module and the measured intermodular transition probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability. The applications of this model to develop cost-effective testing strategies and to determine the expected penalty cost of failures are also discussed. Some future refinements and extensions of the model are presented.

Index Terms—Self-metric software, software reliability, software reliability model, user profile.

I. INTRODUCTION

THE reliability of a system depends on the purpose of the analysis, as well as the method to measure it. In some cases, a hardware system is considered to have failed if any of its components have failed. On the other hand, it has been observed that many systems can continue to provide reliable service in the presence of component failures as long as the service requested is not influenced by the adverse effects of the defective parts. This phenomenon is especially common in software systems. It has been conjectured that every large-scale software system contains some errors. In fact, many of the software reliability models attempt to measure the number of residual bugs in the program [1]-[4]. Yet it is our experience that many of these systems give us "reliable" services. From a user's point of view, the reliability of the system can be measured as the probability that when a user demands a service from the system, it will perform to the satisfaction of the user. Since a large-scale system can perform a variety of services, this measure of reliability has to take into account both the inherent quality of the system (reliability with respect to different services) and the probabilistic distribution of service requests.

In this paper, we will discuss the definition of user-oriented reliability and its relationship to the user profile. A Markov reliability model is formulated under the assumptions that both module reliabilities and intermodule control transfers are independent. The use of this model for sensitivity analysis to identify critical modules is developed. The potential applications of the model to reliability estimation, testing strategy, maintenance philosophy, and estimation of penalty cost are presented. It seems that the concept of user-oriented reliability and a similar reliability model will also be applicable to hardware systems.

II. SOFTWARE RELIABILITY

It is very difficult to give a formal definition of the term "software reliability." One can say that the reliability of a program is equal to one if correct, and zero if incorrect. However, many such "incorrect" programs give us the correct answer most of the time. It seems appropriate for us to evaluate the reliability of the program by a probabilistic measure as one minus the probability of failure. A failure is said to occur if, given the input values and specifications of the computations to be performed by the program, the output values are either incorrect or indefinitely delayed. With such an understanding, and neglecting the performance requirements for the time being, the reliability of a piece of software may be evaluated from two points of view [5]-[7].

The first approach attempts to measure the reliability of a program as it is. We can rate the reliability of a program by the "number" of software bugs left in the program at a particular stage [1]-[4]. It is difficult to count the number of bugs directly. In most systems, the number of "known" bugs is always zero since a bug will be "corrected" after it is discovered. It has been suggested that the number of residual software bugs still in a program can be projected from the rate at which software errors are detected [1]-[4]. Bug counting is a reasonable figure of merit from the system designer's point of view since it measures how close to "perfection" the system is. Using this approach, reliability is a measure of the quality of the program independent of the way the program is used. This may not be an interesting figure of merit from a user's point of view. Bugs in a program are sources of operational failures for the user. The user may be more interested in the opera-
tional reliability of the system than in the number of causes of failures.

Alternatively, we may also treat the reliability of a program from the viewpoint of the quality of the service it provides to a user. The user-oriented reliability of a program (in a certain user environment) is defined as the probability that the program will give the correct output with a typical set of input data from that user environment. Since the sequence of codes executed in a particular run is dependent on the input data, and an error in the nonexecuted statements or branches does not have any effect on the output of the program, the system reliability depends on the probability that a bug is activated in the run. The reliability of the system, therefore, depends on the user profile, which summarizes the dynamic characteristics of a typical execution of the program in a particular user environment. The concept of the user profile is not really new. It has been used in research work in memory management. The address trace is usually used as an effective user profile to develop memory management techniques [8], [9]. Experiments performed have shown that this user profile is remarkably consistent in a particular user environment [10]. If the user environment is not homogeneous, several user profiles may be needed, one for each application of the program. In terms of software reliability, there are many possible measures of the user profile, depending on the formulation of the reliability model. For example, if a telephone system has a typical service reliability objective of 99.98 percent, at least 99.98 percent of all telephone calls should be handled correctly. However, different telephone calls activate different features: local two-party call, three-way call, conference call, call waiting, call transfer, call pick-up, toll call, etc. The probabilistic distribution of the utilization of these features depends heavily on the community served by the telephone system, e.g., business, residential, hotel-motel, etc. A user profile can be defined as the use frequency distribution of these features. A business community will have a different user profile from a residential community. A reliability model may be formulated at this level by measuring the reliability of the different features and using the user profile as a weighting function to obtain system service reliability. Using this measure, one should expect that the system reliability in a business community may be different from that in a residential community. Instead of treating the system as a black box, we can refine the model by considering that these features share hardware and software components. We can then express the system reliability as a function of the reliability of the components and frequency distribution of utilization of these components.

Using this approach, the reliability of the system is a function of both the deterministic properties of the structure of the program and the stochastic properties of the component failure behavior and the utilization of these components by the user. Ideally, we would like a reliability model where the system structure in terms of its components can be easily constructed, where the component reliabilities can be independently determined, and where a simple user profile can be measured easily by monitoring the dynamic behavior of the program. In this paper, a user-oriented software reliability model will be developed to measure the quality of service a program provides to a user. The Markov reliability model uses the program flow graph to represent the structure of the system so that the structure can be easily obtained by analyzing the code. It uses functional modules as the basic components so that the component reliabilities are reasonably independent. It uses the branching characteristics among modules as the user profile so that they can be easily measured in the operational environment. Similar structural models have been proposed by Littlewood [11], [12] to analyze the failure rate of a program and by Booth [13], [14] to analyze software system performance.

III. A USER-ORIENTED SOFTWARE RELIABILITY MODEL

A. Introduction

Using the user-oriented definition of software reliability, let us express the reliability of a program as a function of its components. Since it is the dynamic process generated by the user input that performs the desired computations during a run of the program, a good choice of the user profile is the probabilistic distribution of the set of processes that can be generated by the program in an application environment. Let \( L \) be the set of processes \( P_i \) that can be generated by the program corresponding to different input values. Let \( r_i \) be a random variable such that

\[
    r_i = \begin{cases} 
    1 & \text{if the process } P_i \text{ generates the correct program output} \\
    0 & \text{otherwise.} 
    \end{cases}
\]

Let \( q_i \) be the probability that \( P_i \) will be generated in a user environment. The values of \( q_i \), therefore, define the user profile. The reliability \( R \) of the program can be computed from

\[
    R = \sum_{\forall P_i \in L} q_i r_i.
\]

In order to improve the reliability of a program effectively, one has to improve the reliability of the processes that are executed most often. Notice that \( r_i \) is an inherent property of the program, while \( q_i \) depends on the user profile.

In a large-scale program, the number of processes \( P_i \) can be extremely large. Therefore, it is infeasible for us to evaluate the correctness variable \( r_i \) and measure the probability of occurrence \( q_i \) for each process \( P_i \). In the following section, a simple Markov model is proposed by taking advantage of the modular structure of large-scale programs and using the inter-module transition probabilities as the user profile.

B. Development of the Model

Although there is no universally accepted definition of modularization, most programmers would conceive a module as a logically independent component of the system. This usually implies that a module can be designed, implemented, and tested independently. Since a large-scale program always involves a substantial number of programmers working concurrently, a large program can be viewed as a collection of logically independent "modules." Although there are different criteria for defining a module [15], [16], a module is usually defined to perform a particular function. Let us define the reliability of a module as the probability that the module per-
forms the function correctly, i.e., the module produces the correct output and transfers control to the next module correctly. (If, in a particular software system, several “modules” are logically dependent and perform a well-defined function together, we have to combine these as a single module and define its reliability in the same manner.) When a set of user input is supplied to the program, a sequence of modules will be executed. The reliability of the output will depend on the sequence of modules executed and the reliability of each individual module.

We first assume that the reliabilities of the modules are independent. This means that errors will not compensate each other, i.e., an incorrect output from a module will not be corrected later by subsequent modules. Let us define a process as an execution of the program. Since errors do not compensate each other, the system output of the process is correct if and only if the correct sequence of modules is executed and in every instance of module execution, the module produces the correct result. The reliability of a module, in general, is a function of many factors, and the study of the reliability function of a module is beyond the scope of this paper. However, if no modification is made on the modules and the user environment does not change, the reliability function of a module should remain invariant. We will assume that the module reliability functions can be determined.

We next assume that the transfer of control among program modules is a Markov process. This implies that the next module to be executed will depend probabilistically on the present module only and is independent of the past history. This assumption may not be valid for all types of programs. Although the assumption of Markovian behavior of control flow at the instruction level is questionable, experiments performed by researchers in memory management and scheduling have shown that this assumption may be valid at the macroscopic (module) level [17], [18] for a large number of programs. When no modification is made on the modules, the transition probabilities have been shown to be quite consistent for a given user environment [10], [19]. In the discussion of the model, we will treat these probabilities as constants, and they completely characterize the user environment with respect to the reliability model and serve as the user profile.

C. Reliability Model

Let us represent the control structure of the program by a directed graph where every node $N_i$ represents a program module and a directed branch $(N_i, N_j)$ represents a possible transfer of control from $N_i$ to $N_j$. To every directed branch $(N_i, N_j)$ let us attach a probability $P_{ij}$ as the probability that the transition $(N_i, N_j)$ will be taken when control is at node $N_i$. This transition probability represents the branching characteristics of the decision point at the exit point of the module $N_i$. Let $R_i$ be the reliability of node $N_i$.

Without loss of generality, let us assume that the program graph has a single entry node and a single exit node. Let us consider every node in the graph as a state of the Markov process, with the initial state corresponding to the entry node of the program graph. Two states $C$ and $F$ are added as the terminal states, representing the state of correct output and failure, respectively. For every node $N_i$, a directed branch $(N_i, F)$ is created with transition probability $(1 - R_i)$ representing the occurrence of an error in the execution of module $N_i$. Since errors do not compensate each other, a failure in $N_i$ will ultimately lead to an incorrect system output, regardless of the sequence of modules executed afterwards. This phenomenon is represented by the transition to the terminal state $F$. The original transition probability between $N_i$ and $N_j$ is modified into $R_i P_{ij}$, which represents the probability that the execution of module $N_i$ produces the correct result and control is transferred to module $N_j$. For the exit node $N_n$, a directed branch $(N_n, C)$ is created with transition probability $R_n$ to represent correct termination at the exit node. The reliability of the program is, therefore, the probability of reaching the terminal state $C$ from the initial state of the Markov process.

The reliability of the program can be calculated from the following procedure. Let $\{N_1, N_2, \ldots, N_n\}$ be the set of nodes in the program graph with $N_1$ the entry node and $N_n$ the exit node. Let $R_i$ be the reliability of node $N_i$ and $P_{ij}$ the transition probability of the branch $(N_i, N_j)$. Let $P_{ij} = 0$ if the branch $(N_i, N_j)$ does not exist. The states of the Markov model are $\{C, F, N_1, N_2, \ldots, N_n\}$. Let the transition matrix be $\hat{P}$ where $\hat{P}(i, j)$ represents the probability of transition from state $i$ to state $j$ in the Markov process.

$$
\hat{P} = \begin{bmatrix}
C & F & N_1 & N_2 & \cdots & N_j & \cdots & N_n \\
C & 1 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\
F & 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\
N_1 & 0 & 1 - R_1 & 0 & R_1 P_{12} & \cdots & R_1 P_{1j} & \cdots & R_1 P_{1n} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
N_i & 0 & 1 - R_i & 0 & R_i P_{i2} & \cdots & R_i P_{ij} & \cdots & R_i P_{in} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
N_{n-1} & 0 & 1 - R_{n-1} & 0 & R_{n-1} P_{(n-1)2} & \cdots & R_{n-1} P_{(n-1)j} & \cdots & R_{n-1} P_{(n-1)n} \\
N_n & R_n & 1 - R_n & 0 & 0 & \cdots & 0 & \cdots & 0
\end{bmatrix}
$$
Let $Q$ be the matrix obtained from $\hat{P}$ by deleting all the rows and columns correspond to the absorbing states $C$ and $F$. It is not difficult to show that

\[ S = W^{-1} = (I - Q)^{-1}. \]  

For any positive integer $n$, let the $n$th power of $\hat{P}$ be $\hat{P}^n$. Evidently, $\hat{P}^n(i, j)$ is the probability that starting from state $i$, the chain enters the absorbing state $j \in \{C, F\}$ at or before the $n$th step. The reliability of the program $R$ is the probability of reaching state $C$ (correct termination) from the initial state $N_1$. Hence, we have

\[ R = \hat{P}^n(N_1, C). \]  

Let $S$ be an $n \times n$ matrix such that

\[ S = I + Q + Q^2 + Q^3 + \cdots = \sum_{k=0}^{\infty} Q^k. \]  

If $Q$ is finite, which is the case here, and we let $W = I - Q$, it can be shown that [20]

\[ R = S(1, n) R_n. \]  

**D. An Illustrative Example**

The determination of the reliability of a program using the user-oriented reliability model can be illustrated by the following example. Let Fig. 1 be the directed graph representing the control structure of a program with ten modules where $N_1$ represents the input module and $N_{10}$ is the output module. To simplify our computation, let the reliability of the modules be constants as follows:

- $R_1 = 0.999$
- $R_6 = 0.995$
- $R_2 = 0.980$
- $R_7 = 0.985$
- $R_3 = 0.990$
- $R_8 = 0.950$
Let the branching probabilities $P_{i,j}$ between the modules $N_i$ and $N_j$ be

\[
P_{1,2} = 0.60 \quad P_{1,3} = 0.20 \quad P_{1,4} = 0.20
\]

\[
P_{2,3} = 0.70 \quad P_{2,5} = 0.30
\]

\[
P_{3,8} = 1.00
\]

\[
P_{4,5} = 0.40 \quad P_{4,6} = 0.60
\]

\[
P_{5,7} = 0.40 \quad P_{5,8} = 0.60
\]

\[
P_{6,3} = 0.30 \quad P_{6,7} = 0.30 \quad P_{6,8} = 0.10 \quad P_{6,9} = 0.30
\]

\[
P_{7,2} = 0.50 \quad P_{7,9} = 0.50
\]

\[
P_{8,4} = 0.25 \quad P_{8,10} = 0.75
\]

\[
P_{9,8} = 0.10 \quad P_{9,10} = 0.90.
\]

The program graph is modified into the Markov user-reliability model in Fig. 2. The transition matrix $\hat{P}$ of the Markov model is

\[
\hat{P} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0.001 & 0 & 0.5994 & 0.1998 & 0.1998 & 0 & 0 & 0 & 0
0 & 0.02 & 0 & 0 & 0.6860 & 0 & 0.2940 & 0 & 0 & 0
0 & 0.01 & 0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0
0 & 0.03 & 0 & 0 & 0 & 0 & 0 & 0.388 & 0.5820 & 0
0 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0.38 & 0.57 & 0
0 & 0.005 & 0 & 0 & 0.2985 & 0 & 0 & 0 & 0 & 0.995 & 0.995 & 0
0 & 0.015 & 0 & 0.4925 & 0 & 0 & 0 & 0 & 0 & 0.4925 & 0
0 & 0.05 & 0 & 0 & 0 & 0 & 0.2375 & 0 & 0 & 0 & 0 & 0.7125
0 & 0.025 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.975 & 0.8775
0.9850 & 0.0150 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

and

\[
Q = \begin{bmatrix}
0 & 0.5994 & 0.1998 & 0.1998 & 0 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0.6860 & 0 & 0.2940 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0.388 & 0.5820 & 0 & 0 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0.38 & 0.57 & 0 & 0
0 & 0 & 0 & 0.2985 & 0 & 0 & 0 & 0.2985 & 0.995 & 0.995 & 0.995 & 0
0 & 0 & 0 & 0 & 0.4925 & 0 & 0 & 0 & 0 & 0.4925 & 0
0 & 0 & 0 & 0.2375 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7125
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.975 & 0.8775 & 0 & 0
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
S(1, 10) = [1 - Q]^{-1} (1, 10) = 0.8426.
\]

Also,

\[
R = S(1, 10) R_{10} = (0.8426)(0.9850) = 0.8299.
\]

The reliability of the program is, therefore, 82.99 percent.

In most programs, the matrix $I - Q$ is a sparse matrix so that its inverse can be computed relatively efficiently. This example also shows that the components have to be extremely reliable in order to produce an acceptable system.

E. Sensitivity Analysis

Besides getting an estimation of the quality of a system, this model indicates to the user how to improve the system reliability most effectively by evaluating the sensitivity of the system reliability with respect to that of a module. In the example just given, suppose that $R_5$ is increased from 0.95 to 0.999; the reliability of the system will be increased from 0.8299 to 0.8859. Instead, if $R_8$ is increased from 0.95 to 0.999, the overall reliability is only increased to 0.8661. This illustrates that the system reliability can be more effectively improved by improving module $N_5$ rather than $N_8$.

The sensitivity $s_i$ of the system reliability $R$ with respect to the reliability $R_i$ of module $N_i$ is defined as the partial derivative of $R$ with respect to $R_i$.

\[
s_i = \frac{\partial R}{\partial R_i}.
\]

Since $R = S(1, n) R_n$ and $S = (I - Q)^{-1} = W^{-1}$, it can be shown that [21]

\[
R = R_n(-1)^{n+1} \frac{|M_n|}{|W|}
\]

(9)
where $|M_{n1}|$ is the minor of $W(n, 1)$ and can be evaluated as the determinant of the remaining matrix by removing the $n$th row and first column of $W$ and $|W|$ is the determinant of the square matrix $W$.

Let us now try to express $R$ as a function of $R_i$. Recall that the branching probabilities $P_{ij}$ are independent of the module reliabilities. The determinant $|W|$ can be evaluated by expanding it along any row $i$ in terms of the cofactors

$$|W| = W(i, 1) \alpha_{i1} + W(i, 2) \alpha_{i2} + \cdots + W(i, n) \alpha_{in}$$

(10)

where $\alpha_{ij}$ is the cofactor of $W(i, j)$.

Since the module reliabilities are independent, the cofactor $\alpha_{ij}$ is not a function of $R_i$. Therefore,

$$|W| = K_{i1} + K_{i2} R_i$$

(11)

where

$$K_{i1} = \alpha_{i1},$$

$$K_{i2} = -\sum_{j=1}^{n} P_{ij} \alpha_{ij},$$

and $K_{i1}$ and $K_{i2}$ are not functions of $R_i$.

Similarly, the determinant $|M_{n1}|$ can also be evaluated by expanding it along any row $i$ in terms of the cofactors $B_{ij}$ of $M_{n1}(i, j)$,

$$|M_{n1}| = k_{i1} + k_{i2} R_i$$

(12)

where

$$k_{i1} = \begin{cases} 0 & \text{for } i = 1 \\ B_{i(i-1)} & \text{for } i = 2, \cdots, n - 1. \end{cases}$$

$$k_{i2} = -\sum_{j=2}^{n} P_{ij} B_{i(j-1)}$$

and $k_{i1}$ and $k_{i2}$ are not functions of $R_i$.

$$R = (-1)^{n+1} R_n \frac{k_{i1} + k_{i2} R_i}{K_{i1} + K_{i2} R_i} \quad \text{for } i = 1, \cdots, n - 1.$$

(13)

The sensitivity can be computed by differentiating (13) with respect to $R_i$,

$$S_i = \frac{\partial R}{\partial R_i} = (-1)^{n+1} R_n \frac{k_{i1} k_{i2} - K_{i1} k_{i2}}{(K_{i1} + K_{i2} R_i)^2}$$

(14)

$$S_n = \frac{\partial R}{\partial R_n} = \lambda (1, n)$$

(15)

where $k_{i1}$, $k_{i2}$ and $K_{i1}$ and $K_{i2}$ are defined in (12) and (11) and can be evaluated from the matrix $I - Q$.

### IV. Applications

By measuring the operational characteristics of a piece of software, the user-oriented reliability model gives us two types of figures of merit: the reliability of its service and the sensitivity of its reliability with respect to that of different modules. The reliability measure gives the user a direct estimation of the confidence he should place in the program. It also indicates the quality of service the program is going to provide and whether further testing and improvement are required for his application. The sensitivity coefficients with large values indicate to the user the critical modules which have the greatest impact on system reliability.

The reliability model can be used to develop an effective testing strategy given limited testing resources. Testing techniques are usually designed to show that the program is as close to a "correct" program as possible by detecting as many bugs as possible. A more effective testing strategy is to show that the program is as "reliable" as possible (for a given testing budget) by concentration on the module with the largest sensitivity coefficients. The reliability model identifies the critical modules where proof of correctness or exhaustive testing techniques should be used. Stricter acceptance tests should be imposed on these modules. For example, if a path sensitization strategy is used for testing, it may be sufficient to cover all decision-to-decision paths on noncritical modules, but necessary to cover all simple paths in addition to loop boundary conditions of critical modules. Module reliability at execution time can also be improved by incorporating self-checking and recovery capabilities [22], [23]. The sensitivity analysis will determine where such fault-tolerant capabilities should be introduced most effectively. For example, in a telephone system, audits have been used to improve the operational reliability of the system. The user-oriented reliability model will enable us to distribute our design effort as well as our real-time processing resources to audit the most critical data structures. On the other hand, we should also consider the cost of implementing these reliability improvements which may depend on such factors as size, logical complexity and structure, relation-
ship with other modules, and the programmer's understanding of the module. Therefore, the distribution of our resources should depend both on the effectiveness (sensitivity coefficient $s_j$) and the marginal cost of improving the reliability of the module.

With respect to program maintenance, the sensitivity analysis indicates modifications that may have a traumatic effect on the reliability of the system. If critical modules have been shown to be reliable, one should avoid as much as possible modifying them when considering maintenance alternatives (of course, this does not preclude modifications that will improve their reliability). One must be aware that correcting a non-critical error may introduce a bug to the critical modules, thus degrading the system reliability more than improving it. The priority of different requests for correcting errors in modules should be arranged in the order of their impact on the improvement of the system reliability, as measured by their sensitivity coefficients.

An extension of the model can also enable the user to evaluate the effect of an error in a module. It is observed that not all software bugs are equally costly. A software bug in the missile firing module for a defense system may make the whole program unacceptably unreliable, even when the rest of the program is error-free. A software bug which can cause the loss of a master file in a business transaction system is also more critical than one which causes an error in a single transaction. Therefore, besides the frequency of execution, the importance of the reliability of a software module should also take into account the criticality and penalty cost of a bug in that module. If the effect of an error in a module is known, the Markov reliability model can be extended by attaching a penalty cost $C_i$ to the branch from node $N_i$ to the failure state $F$ as the penalty cost of a software bug in that module. Let $X_i$ be the expected penalty cost given that the program control is now at node $N_i$. The expected penalty cost of the program is, therefore, $X_1$, which can be solved from the following set of $n$ recurrent equations:

$$X_i = C_i(1 - R_i) + R_i \sum_{j=1}^{n} P_{ij} X_j \quad \text{for} \quad i = 1, \cdots, n. \quad (16)$$

An evaluation of the expected penalty cost of the program may be a better criterion for the acceptance of the system than the absolute reliability of the program. Similarly, a sensitivity analysis can also be performed to determine the critical modules which have the most significant effect on the expected penalty cost of the program. The penalty cost is, therefore, another figure of merit in evaluating the criticality of a module.

V. ESTIMATION OF PARAMETER VALUES

The parameters of the reliability model are the transition probabilities $P_{ij}$ and the reliability of the individual modules $R_i$. In the model, we assume that the system will enter the terminal failure state $F$ when there is an error in the module. The transition probabilities, therefore, represent the branching characteristics of the program when the modules are functioning correctly. The branching characteristics can be measured by selecting a large representative sample of $N$ sets of valid input data, running the program, and measuring the frequency of execution of each intermodule transfer. A technique for optimal instrumentation of the program to measure these frequencies using self-metric software can be found in [24] and [25].

The reliability of each module may be estimated using a variety of methods. It may be determined by stochastic testing, i.e., by selecting a representative sample of $X$ sets of valid input data from a user environment, running the program, and finding the number $Y$ of correct sets of program output. The reliability of the program is estimated to be $Y/X$. When the number of tests is large, a reasonable estimation can be obtained. If the system is already operational, the reliability of each module may also be projected from its operational history, i.e., by measuring the ratio of the number of times it produced correct output to the number of times it was used. In recent years, many models have also been formulated to predict system reliability using the "black box" approach [26], [27]. These models may also be used to estimate the reliability of a module.

VI. CONCLUSIONS

In this paper, the concepts of user-oriented reliability and user profile have been presented. The reliability of a system is expressed as a function of the reliabilities of its components and the user profile. A Markov model is developed under the assumptions of independence of module reliability and the Markovian behavior of control transfer among modules. The applicability of the model to a particular program in a particular user environment depends on the validity of these assumptions. Although there are many criteria used in decomposing a system into modules, modules are usually designed so that the function of a module is independent of the source of its input, the destination of its output, and the past history of the module being used. Since these modules can be independently implemented, separately tested, and reused from different places, it is reasonable to conjecture that the reliabilities of the modules are independent. This assumption has been shown to be valid by an experiment done by Parnas [28] on a system in which the modules are well defined in terms of external characteristics [15], [29]. The Markovian behavior of control transfer may be applicable only to certain systems. In some business transaction systems and telephone switching systems, the system has no loops and no locality behavior has been observed. The sequence of operations executed depends mainly on the transaction of feature that the user requests. For a particular transaction or feature, the branching characteristics of the system seem to be quite independent of the values of input data variables in rather large domains. The same properties have been observed in many programs, and form the basis of many memory management strategies through measurement of program behavior [30].

The user-oriented reliability model enables us to evaluate the reliability of the service of the system and the sensitivity of system reliability with respect to component reliabilities. Using this model, we can predict the operational quality of the system and determine how much testing will be sufficient for
a particular reliability performance goal of an application. The sensitivity analysis can indicate to us the program modules most critical to system reliability so that exhaustive testing and run-time fault-tolerance capabilities such as audits can be concentrated on them. It indicates that perhaps the goal of testing should be oriented towards showing reliability instead of correctness. If the penalty cost of a failure can be estimated, the expected penalty cost may be a better figure of merit than reliability.

The user-oriented reliability model also raises some interesting questions about system design. The model indicates that program modules used most often during execution time probably are the critical module from a reliability point of view. We would like to keep their structures simple and elegant, perhaps sacrificing efficiency. However, they are also the modules where efficiency is most important due to the frequency of use. We may want to optimize them, sacrificing simplicity and structure. Can we design a system that is both reliable and efficient? A reasonable answer seems to be: optimize only after you know it is correct, i.e., 100 percent reliable. This indicates that either we have to optimize code without sacrificing structure or we can only optimize small parts only where correctness can be shown.

The applicability of user-oriented reliability is not limited to software alone. Hardware reliability can also be estimated by the number of "users" being affected by a failure. For example, the maintenance strategy of a telephone system may be oriented towards minimizing the number of phone calls affected. Fault resolution and the frequency of periodic diagnosis may be tailored according to the severity of the fault. Down time may be measured in virtual time in terms of the expected number of phone calls denied service rather than physical time. As more and more logic is integrated into a chip, the possibility of a design error may be just as great as in software. The proposed user-oriented software reliability model may be refined to account for component failures due to aging in addition to design errors. Hopefully, a practical reliability model for both software errors and hardware faults can be formulated in the future.

REFERENCES


