A USER-ORIENTED SOFTWARE RELIABILITY MODEL

ROGER C. CHEUNG

Bell Telephone Laboratories,
Naperville, Illinois 60540

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 ulitization of the components to provide the service. In this paper a user-oriented 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 is 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 inter-modular 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 failure are also discussed. Some future refinements and extensions of the model are presented.

INTRODUCTION

The reliability of a system depends on the purpose of the analysis as well as the method to measure it. In many 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 so 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.[1,2,3] 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 inter-module control transfers are independent. The use of this model for sensitivity analysis to identify critical modules is developed. The application of the model to reliability estimation testing strategy, maintenance philosophy and estimation of penalty cost are presented. The concept of user-oriented reliability and the reliability model are applicable to both hardware and software systems.

SOFTWARE RELIABILITY

It is very difficult to give a formal definition to 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 use 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 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.[4,5]

The first approach attempts to measure the reliability of a program directly. We can rate the reliability of a program by the "number" of software bugs in the program, i.e., the number of mistakes made during the design and implementation of the program.[1,2,3] This number may then be used to determine the mean time to failure of the program. It has been suggested that the rate at which software errors are detected could be used as a projection of the number of software bugs still in the program [1,2,3]. Using this approach, reliability is an inherent property of the piece of software independent of the way the program is used.

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 desired output with a typical set of input data from that user environment. Since the sequence of codes executed 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 software system reliability depends on the probability that a bug is activated in a typical run. The reliability of the system depends on the "component" reliabilities weighted by the user profile, which summarizes the dynamic characteristics of a typical execution of the program in a particular environment. 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 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 2-way, 3-way, conference call, call waiting, call transfer, call pick-up, toll call, etc. The 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 frequency distribution of use 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 weighing 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. The model can be refined by considering that these features share hardware and software components. We can then express the system reliability as a function of the reliability and frequency distribution of utilization of these components.

Ideally we would like a reliability model where the "component" reliabilities can be independently determined. We would also like to have a simple user profile which can be measured easily by monitoring the dynamic behavior of the program. 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[6,7]. Experiments performed have shown that this user profile is remarkably consistent in a particular user environment[8]. If the user environment is not homogeneous, several user profiles may be needed, one for each application of the program.

A USER-ORIENTED SOFTWARE RELIABILITY MODEL

Development of the model

A large program can be viewed as a collection of logically independent modules, which can be designed, implemented and tested independently . Although there are different criteria for defining a module[9,10], a module is usually defined to perform a particular function. Let us define the reliability of a module as the probability that the module performs 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[11,12] 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[8,13]. 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.

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 occurence 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 $\hat{P}$, as shown in (1), where P (i,j) represents the probability of transition from state i to state j in the Markov process. Let Q be the matrix obtained from $\hat{P}$, be deleting all the rows and columns corresponding to the absorbing states C and F. 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 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

$$R = \hat{P}^n (N_1, C) \tag{2}$$

Let S be an n by n matrix such that

$$S = I + Q + Q^2 + Q^3 + \ldots = \sum_{k=0}^{\infty} Q^k$$

If Q is finite, which is the case here, and let W = I- Q, it can be shown that [14]

$$S = W^{-1} = (I-Q)^{-1} \tag{3}$$

It is not difficult to show that

$$R=S(1,n)R_n \tag{4}$$

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 evaluation the sensitivity of the system reliability with respect to that of a module.

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.

$$s_i = \frac{\partial R}{\partial R_i} \tag{5}$$

Since $R = S(1,n) R_n$ and $S = (I-Q)^{-1} = W^{-1}$, it can be shown that [15]

$$R = R_n (-1)^{n+1} \frac{|M_{n1}|}{|W|} \tag{6}$$

$$\hat{P} = \begin{array}{c} \\ C \\ F \\ N_1 \\ \vdots \\ N_i \\ \vdots \\ N_{n-1} \\ N_n \end{array} \begin{array}{c} \begin{array}{cccccccc} C & F & N_1 & N_2 & \cdots & N_j & \cdots & N_n \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1-R_1 & 0 & R_1P_{12} & \cdots & R_1P_{ij} & \cdots & R_1P_{1n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 1-R_i & 0 & R_iP_{i2} & \cdots & R_iP_{ij} & \cdots & R_iP_{in} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 1-R_{n-1} & 0 & R_{n-1}P_{(n-1)2} & \cdots & R_{n-1}P_{(n-1)j} & \cdots & R_{n-1}P_{(n-1)n} \\ R_n & 1-R_n & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \end{array} \tag{1}$$

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 1st column from 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,

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

where α_{ij} is the cofactor of W (i,j).

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

$$|W| = K_{1i} + K_{2i} R_i \quad \text{for } i = 1,\ldots, n-1, \quad (7)$$

where $K_{1i} = \alpha_{ii}$,

$$K_{2i} = -\sum_{j=1}^{n} P_{ij}\alpha_{ij},$$

and K_{1i} and K_{2i} 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),

$$|M_{n1}| = k_{1i} + k_{2i}R_i \quad \text{for } i = 1,\ldots, n-1. \quad (8)$$

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

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

and k_{1i} and k_{2i} are not functions of R_i.

$$R = (-1)^{n+1} R_n \frac{k_{1i} + k_{2i}R_i}{K_{1i} + K_{2i}R_i} \quad \text{for } i=1,\ldots,n-1 \quad (9)$$

The sensitivity can be computed by differentiating equation (9) with respect to R_i,

$$s_i = \frac{\partial R}{\partial R_i} = (-1)^{n+1} R_n \frac{K_{1i}k_{2i} - K_{2i}k_{1i}}{(K_{1i} + K_{2i}R_i)^2} \quad (10)$$

for i = 1, ...,n-1

$$s_n = \frac{\partial R}{\partial R_n} = S(1,n)$$

where k_{1i}, k_{2i} and K_{1i} and K_{2i} are defined in equations (8) and (7) and can be evaluated from the matrix I-Q.

APPLICATION

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 is 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 non-critical 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[16,17]. 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 processor 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 factors as size, logical complexity and structure, relationship 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_i) 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:

$$X_i = C_i(1-R_i) + R_i \sum_{j=1}^{n} P_{ij} X_j \quad \text{for } i = 1,\ldots, \quad (11)$$

An evaluation on the expected penalty cost of the program may be a better criterion for the acceptance of the system than the absolute reliabilty 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.

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 inter-module transfer. A technique for optimal instrumentation of the program to measure these frequencies using self-metric software can be found in Reference.[18,19]

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.[20,21] These models may also be used to estimate the reliability of a module.

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 usally 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 use of the module. 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[22] on a system in which the modules are well-defined in terms of external characteriscs[9,23]. The Markovian behavior of control transfer may be applicable only to certain systems. In some business transaction systems and telephone switching systems, the program has no loops and no locality behavior has been observed. The sequence of operations executed depends mainly on the transaction or feature that the user requests. For a particular transaction or feature, the branching characteristics of the system seems 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.[24]

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 on 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 develop techniques to optimize code without sacrificing its structure or we can optimize only the small parts 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.

Acknowledgment Part of the research was supported by Rome Air Development Center, Contract F30602-76-C-0397, while the author was at Northwestern University.

REFERENCES

[1] Shooman, M. L. "Operational Testing of Software Reliability During Program Development," Record of IEEE Symposium Computer Software Reliability, New York City, 1973.

[2] Jelinski, Z. and Moranda, P. B. "Software Reliability Research," Statistica Computer Performance Evaluation, edited by Walter Freiberger, Academic Press 1972, p. 464.

[3] Moranda, P. B. "Predictions of Software Reliability During Debugging," 1975 Proceedings of the Annual Reliability and Maintainability Symposium, Jan. 1975, Washington D.C.

[4] Ramamoorthy, C. V., Cheung, R. C., and Kim, H. H. "Reliability and Integrity of Large Computer Programs", Computer System Reliability, Infotech State of a Art Report 20, 1974.

[5] Cheung, R. C. A Structural Theory for Improving Software Reliability, Ph.D. Thesis, Dept. of EECS, Univ. of California, Berkeley, California, 1974.

[6] Belady, L. A. "A Study of Replacement Algorithms for a Virtual-storage Computer", IBM Systems Journal, Vol. 5, No. 2, 1966, pp. 78-101.

[7] Mattson, R. L., Gecsei, J., Slutz, D. R., and Traiger, I. L. "Evaluation Techniques for Storage Hierarchies", IBM System Journal, Vol. 9, No. 2, 1970, pp. 78-117.

[8] Hatfield, D. J. "Experiments of Page Size, Program Access, Patterns, and Virtual Memory Performance", IBM Journal of Res. Develop., Jan. 1972, pp. 58-66.

[9] Parnas, D. L. "On the Criteria To Be Used in Decomposing Systems into Modules", Comm. ACM, Vol. 15, No. 12, Dec. 1972, pp. 1053-1058.

[10] Myers, G. J. Reliable Software Through Composite Design, Petrocelli/Charter, New York 1975.

[11] Ramamoorthy, C. V. "The Analytic Design of A Dynamic Look Ahead and Program Segmenting System for Multiprogrammed Computers", Proc. ACM Nat. Conf., 1966, pp. 229-239.

[12] Pinkerton, T. B. "Program Behavior and Control in Virtual Storage Computer Systems", Tech. Rep. 4, Univ. of Michigan, 1968.

[13] Haftield, N. J. and Jerald, J. "Program Restructuring for Virtual Memory", IBM Syst. J., Vol. 10, No. 3, 1971, pp. 168-192.

[14] Cinlar, E. Introduction of Stochastic Processes, Prentic-Hall, 1975, Chap. 5.6.

[15] Nomizu, K. Fundamentals of Linear Algebra, McGraw Hill, 1966.

[16] Randell, B "System Structure for Software Fault-tolerance", Proc. 1975 Int. Conf. on Reliable Software, 1975 pp. 437-457.

[17] Yau, S. S. And Cheung, R. C. "Design of Self-Checking Software", Proc. 1975 Int. Conf. on Reliable Software, 1975, pp. 450-457.

[18] Cheung, R. C. Kim H. H., Ramamoorthy, C. V., and Reddi, S. S. "Automated Generation of Self-metric Software", 7th Hawaii International Conference on System Sciences, Jan. 1974.

[19] Cheung, R. C. and Ramamoorthy, C. V. "Optimal Measurement of Program Path Frequencies and its Applications", Proc. of 1975 International Federation of Automatic Control Congress, Aug. 1975.

[20] Brown, J. R. and Lipow, M. "Testing for Software Reliability," Proc. of 1975 Int. Conf. on Reliable Software, April, 1975, pp. 518-527.

[21] Nelson, G. C. Software Reliability, TRW-SS-75-05, Nov. 1975.

[22] Parnas, D. L. "Some Conclusions from an Experiment in Software Engineering Techniques", Fall Joint Comp. Conf., 1972, pp. 325-329.

[23] Parnas, D. L. "A Technique for Software Module Specifications with Examples", Comm. ACM, May 1972, pp. 330-336.

[24] Special issue on Measurement of Program Behavior, Computer, Nov. 1976.