Use of Importance Sampling and Related Techniques to Measure Very High Reliability Software[1]

Myron Hecht and Herbert Hecht
SoHaR Incorporated
8421 Wilshire Blvd., Suite 201
Beverly Hills, CA 90211
myron@sohar.com, herb@sohar.com
323-653-4717

Abstract—Computer-based control systems have grown more complex over the past two decades. Thus, the software aspects of system reliability are an increasingly important concern. Current methods of software and system reliability prediction – whether measurement based or incorporating reliabilty growth models – can not accurately predict failure rates of greater than 10^{-6} per mission hour. This paper describes a new methodology for more accurately predicting failure rates of very high reliability systems. The methodology enhances conventional measurement-based reliability assessment with a method incorporating the results of stress testing called importance sampling. By means of importance sampling in conjunction with a system model, acceleration factors can be associated with stress testing much as is currently done with elevated temperature life testing of hardware components.

TABLE OF CONTENTS

1. INTRODUCTION
2. RELATED RESEARCH
3. OVERVIEW OF IMPORTANCE SAMPLING
4. ASSESSMENT OF VERY HIGH RELIABILITY SOFTWARE
5. EXAMPLES
6. CONCLUSIONS

1. INTRODUCTION

The most common method for achieving high reliability software is to put standards on the development process. An example of such a paradigm is the Capability Maturity Model (CMM) developed by the Software Engineering Institute at Carnegie Mellon University. However, evidence that such process-based methods yield more reliable high integrity systems is anecdotal at best, despite the attempts of many investigators over many years. Ideally, it would be desired to actually measure the reliability and safety of high integrity systems. Unfortunately, using conventional methods of test, probabilistic modeling, and statistical analysis, it is quite difficult to establish failure rates of less than 10^{-6} per mission hour [Butler89 ,Littlewood89, Butler93, Hecht96]. For mission and life-critical systems, much lower failure rates must be established.

This paper describes a methodology for the empirical quantitative verification reliability in high reliability digital systems. The methodology enhances measurement-based reliability assessment with a method of incorporating the results of stress testing called importance sampling. The basis of the method is that by emphasizing the testing of software in the unusual modes of operation (extreme system configurations, failure/recovery testing, etc.), it is possible to obtain data that represent many more years of operational experience than the actual test time. By means of importance sampling in conjunction with a system model, acceleration factors can be associated with stress testing much as is currently done with elevated temperature life testing of hardware components. The measurement of very low failure probabilities through the quantitative, measurable, and repeatable results of testing reduces the uncertainty in the development and certification of software based high reliability and integrity systems. The benefit is a reduction of development cost and schedule.

.

2. RELATED RESEARCH

This section describes previous work on reliability assessment methods including reliability growth models, measurement-based dependability, and stress testing.

2.1 Reliability Growth Models

Research on quantification of software reliability started in the early 1970's [Farr96] with reliability growth models that attempt to predict the future software failure based on extrapolation of trends in its past performance. Many reliability growth models have been proposed to characterize reliability growth for software under development. These

[1] 0-7803-5846-5/00/$10.00 © 2000 IEEE

models evaluate the reduction in failure frequency during successive test intervals to estimate the software reliability at the conclusion of the test. Some of the recommended models are the Schneidewind model, the generalized exponential model, the Musa/Okumoto Logarithmic Poisson model, and the Littlewood/Verrall model [ANSI92]. Applications of these models have all been demonstrated using real data from software with observed failure rates from 10^{-1} to 10^{-5} per hour [Abdel-Ghaly86, Musa87, Kanoun97].

It has been shown that the quantification of very low failure rate (high reliability) systems is infeasible by using reliability growth models along with traditional testing techniques [Butler93]. An illustrative example in the study is that it would take 10^{8} to 10^{10} hours (thousands of years) of testing to demonstrate a failure rate of 10^{-7} to 10^{-9} per hour assuming one copy of software would be tested and one failure would be observed. Even if 10 copies (which is likely) or 100 copies (which is unlikely) of the software are tested concurrently, it would still take hundreds (for 10 copies) or tens (for 100 copies) of years. The study also cited comments of other experts in the field on this issue, including the following:

> "Clearly, the reliability growth techniques of §2 [a survey of the leading reliability growth models] are useless in the face of such ultra-high requirements. It is easy to see that, even in the unlikely event that the system had achieved such a reliability, we could not assure ourselves of that achievement in an acceptable time." [Littlewood89]

Another limitation of reliability growth models is their lack of ability to model software structure. Reliability growth models treat the software as a black box and assess the reliability of the software as a whole. Critical software systems include fault tolerance mechanisms, such as error detection and handling, redundancy management, and backup tasks. As such, the dependability (reliability, availability, etc.) of the whole software system cannot be simply quantified by observed failures at the component (task) level. For example, a transient task failure may be covered by the fault tolerance provisions and thus, critical functions will not be affected. This scenario has been verified by several studies [Gray90, Lee95, Tang95a] which showed that 80% to 95% of software failures in real-time fault tolerant systems are recoverable using a primary/copy (dual redundant) software architecture. In such a case, reliability growth models are not sufficient for dependability quantification and structured dependability models need to be used.

2.2 Measurement-Based Reliability Assessment

Measurement-based reliability assessment involves the development of system reliability models (reliability block diagram, *k*-out-of-*n* model, and Markov chain) and then using test or operational data to determine the values of the parameters within the model. Reliability models have been used to evaluate operational software based on failure data collected from commercial computer operating systems for over 10 years (e.g., [Hsueh87, Lee93]). The methodology was recently extended to evaluate availability for air traffic control software systems in the late testing phase [Tang95a] and early operational phase with multiple sites [Tang98].

Both reliability diagrams and *k*-out-of-*n* models are combinatorial models and typically assume failure independence among modeled components. Markov chains are stochastic models which can incorporate interactions among components and failure dependence.

However, the current practice of measurement-based evaluation for individual software systems (with the number of installations < 100) is still limited to failure rates of 10^{-2} to 10^{-5} per hour and an availability of three to five 9's (0.999 to 0.99999). For example, the newly developed FAA Voice Switching and Control System (VSCS) is being installed in 21 major U.S. air traffic control centers and the authors were tracking the system operational availability based on failure reports from installed sites during a period between 1995 and 1996. In this work, the system availability (dominated by software) was evaluated to have reached five 9's as of June 30, 1996. If no major failure occurs in the future, it will take 15 years of normal operation for the 21 sites to demonstrate an availability of the required seven 9's at the 80% confidence level, using the upper bound based evaluation method discussed in [Tang95a]. Therefore, in order to quantify availability for systems with such high requirements, it is necessary to explore accelerated testing and assessment methods.

2.3. Stress Testing

A feasible acceleration approach is the stress testing of components or operations which can induce critical failures Stress testing techniques are sometimes used by software manufacturers to test rarely exercised components or operations, such as redundancy management software. A particular stress testing technique, fault injection, has also been used on software in laboratory studies in recent years [Chillarege89, Arlat90, Kao93, Siewiorek93, Avresky96, Voas97b]. But none of these stress testing techniques has been directly connected to reliability or availability quantification. Even if thorough testing has been performed and the software has actually reached a very high reliability, the lack of evaluation means prevents us from knowing what the software reliability is.

The two following events demonstrate the value of stress testing. Both were triggered by data being out of the expected ranges, although causes may differ (software and hardware problems). However, both events could be avoided if the

corresponding data checking and protection code exist in the software. The need for such checking and protection could have been detected by means of a stress test plan which included testing for extreme values within and out of expected ranges.

- *The Ariane 5 launch vehicle destruction* occurring on June 4, 1996 was due to a data error in a computer and the unavailability of the software to handle the error. The faulty data forced the On board computer (OBC) to change the angle of attack that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher. [Voas97a]

- *The Voice Switching and Control System (VSCS)* located at Seattle experienced a Type I failure (a problem that precludes the primary air traffic control system mission objective of controlling aircraft) on August 11, 1995 when all of the eight Air to Ground telephony switch shelves (A/G shelf) failed. The event was widely reported by the media. The problem diagnosis provided by the manufacturer identified the likely root cause as an undetected fault in a memory chip. The fault corrupted the length field (set to 0-length) of a message broadcast by an A/G shelf. All other A/G shelves (including primaries and standbys), upon receiving this invalid message, reset and cleared all application code from their processors simultaneously due to a general protection fault caused by the 0-length. [Tang98]

The key problem is that in the stress testing, the test cases are not very representative of the operational usage. The test cases used represent a biased operational profile and the failure rate measured is greater than the actual failure rate. If the degree of the amplification is known, it could be adjusted back to the actual value. Importance sampling, described in the next section, is a method to statistically assess the value of this acceleration factor.

3. Overview of Importance Sampling

Importance sampling is a statistical method that keeps estimates obtained from the sample correct at a high level of confidence while reducing sampling size[Kahn53]. Specifically, it allows the analysis of rare events with a high degree of accuracy when Monte Carlo techniques are used. The method, summarized below, has been used in recent years to reduce the number of runs in Monte Carlo simulations for evaluating computer dependability [Goyal92, Choi93].

Assume that a random variable X has probability density function (p.d.f.) $f(x)$ and that $Y=h(X)$ is a function of X. Our goal is to estimate the expected value of Y,

$$\theta = E[Y] = E[h(X)] = \int_{-\infty}^{+\infty} h(x)f(x)dx \quad \textbf{(1)}$$

through sampling. That is, we generate a sample $\{x_1, x_2, ..., x_n\}$ according to $f(x)$, therefore generating $\{y_1, y_2, ..., y_n\}$, and then calculate

$$\tilde{\theta} = \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{n}\sum_{i=1}^{n} h(x_i) \quad \textbf{(2)}$$

For most digital control systems involved in aerospace applications, it is very expensive to generate a statistically significant sample of X. For example, if $y_i = h(x_i) = 0$ for most generated x_i, we may need an extremely large size of sample to estimate θ with a high level of confidence. However, if we can make the rare x_i's which are "important" for estimating θ be much more frequently selected in the sampling while keeping the estimate unbiased, the sample size would be greatly reduced. This is the basic idea of the importance sampling method.

In importance sampling, we change the p.d.f. of X from $f(x)$ to $g(x)$ such that those x's which are of importance in our parameter estimation have higher occurrence probabilities in $g(x)$. We use X' to represent the variable which has p.d.f. $g(x)$. By Equation (1), we have

$$\theta = \int_{-\infty}^{+\infty} h(x)f(x)dx = \int_{-\infty}^{+\infty} h(x)\frac{f(x)}{g(x)}g(x)dx \quad \textbf{(3)}$$

$$= \int_{-\infty}^{+\infty} h(x)\Lambda(x)g(x)dx$$

where

$$\Lambda(x) = \frac{f(x)}{g(x)} \quad \textbf{(4)}$$

is called *likelihood ratio*. Let $Y' = h(X)\Lambda(X)$, then Equation (3) becomes

$$\theta = \int_{-\infty}^{+\infty} y' g(x)dx = E[Y'] \quad \textbf{(5)}$$

Thus, instead of sampling from $f(x)$ to estimate the expected value of Y, the experiment is changed to sampling from $g(x')$ to estimate the expected value of Y'. That is, we generate a

sample $\{x'_1, x'_2, ..., x'_n\}$ according to $g(x')$, therefore generating $\{y'_1, y'_2, ..., y'_n\}$, and then calculate

$$\tilde{\theta} = \bar{Y}' = \frac{1}{n}\sum_{i=1}^{n} y'_i = \frac{1}{n}\sum_{i=1}^{n} h(x'_i)\Lambda(x'_i) \qquad \textbf{(6)}$$

The result of equation 6 basically shows that a given failure rate, measured under conditions of stress testing, can be scaled back to an expected value for normal operation if the operational profile is known.

An example of the use of an operational profile is in the stress testing of FASTAR, the AT&T's Fast Automated Restoration Platform. "The FASTAR operational profile showed that cable cuts happen very infrequently. Restoration of cable cuts was certainly the most critical operation of the system. Since it was not feasible to execute the test program for a year to simulate the expected 10 cuts, we designed accelerated test runs. In the case of the Restoration Node Controller, load was accelerated by a factor of 24 during stability tests. This could execute a year's operational profile in the test environment within a matter of weeks." [Musa96]

A second example of the use of an operational profile was the stability test of the VSCS air traffic control system before the first installation of the system in the field. During the 72 hours of the stability test, the operation "switchover" between the primary and standby Air to Ground Communication Switching Subsystems was performed 30 times, while the typical rate of this operation in the field is only once a day [Tang95b]. This translates to an acceleration factor of 10.

An example of the use of an inverted operational profile that represents test cases from ultra-rare regions in fault injection case studies on software from NASA [Voas97c]. In case study 1, Yaw, serving as a small yawdamp controller for a Boeing 737 aircraft delivered to NASA for research purposes, was tested under fault injections. In case study 2, Autopilot, providing the autopilot controls for a Boeing 737 airplane, was tested under fault injections. For the original operational profile, the failure tolerance score is 0.43 for Yaw and 0.95 for Autopilot. For the inverted operational profile, the failure tolerance score is reduced to 0 for Yaw and 0.05 for Autopilot.

Table 1 shows some characteristics of the failure data from the NASA/JPL Deep Space Network (DSN) [Hecht93], where FC denotes frequently executed code and RC denotes rarely executed code. In this program, the RC included redundancy management, exception handling, initialization and calibration routines. Failures in these functions were frequently much more critical to the value of the spacecraft data than failures affecting routine data transmissions. Also, it was estimated that over a period of several months the FC would be exercised at least 100 times as much as the RC. Thus, the execution probability of the FC is over 100 times greater than that of the RC, i.e., P_C (importance factor) is less than 0.01, or an estimated P_{upp} is 0.01.

Several observations can be drawn from this table:

- The fault density measured from test for the RC was only one-third that of the FC. This indicates that the test case distribution attempted to match the operational profile in terms of the FC and RC. By applying importance sampling to the operational profile to increase the number of RC test cases a greater number of failures would have been encountered during test and the removal of these faults would have reduced the number of failures in the operational phase. That is, stress testing of the RC such as redundancy management and exception handling code could achieve the highest benefits.

Table 1. Characteristics of DSN Code

Code characteristic	Frequently Executed Code	Rarely Executed Code
Program size (KSLOC)	185.1	144.3
Number of faults found in test	893	235
Fault density measured from test	4.82	1.63
Failures during first year operation	32	42
Failures during last 4 months of year	9	32

- If the acceleration factor is 100, the 42 RC failures incurred during the first year of operation would probably have been found in a few days of stress testing concentrated on the critical operations. To find the 32 faults responsible for the FC failures during the first operational year would probably have required almost a year of test since these functions were presumably accessed in operation as frequently as they were in test.

- The comparison of the FC and RC failures during the last four months (9 vs. 32) shows that the RC failures were dominant in the well tested software. The dependability estimated from the RC failures would thus be close to the actual dependability (based on all

failures). Even for the first whole operational year, the dependability estimated from the 42 RC failures occurring during the year would be in the same order of that estimated from all the 75 failures of the year.

4. MEASUREMENT-BASED ASSESSMENT OF VERY HIGH RELIABILITY SOFTWARE

Figure 1 shows the layout of an approach to the measurement-based assessment of very high reliability software. The methodology utilizes importance sampling discussed in Section 3 together with the reliability growth, modeling, and stress testing methods discussed in Section 2. The mapping between the operational profile and the stress testing profile is quantified by importance sampling. This quantification allows the estimation of failure rates, which depend on rare events, to be unbiased.

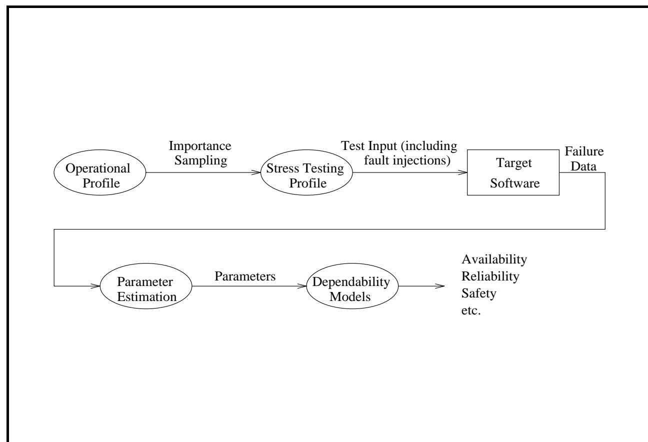

Figure 1 Top Level View of the Approach

The following subsections discuss in greater detail how how methods that can be used to implement the approach. The first subsection discusses determination of the operational profile, the second discusses quantitative analysis of testing results using importance sampling and the operational profile, the third discusses data collection, and the fourth discusses parameter estimation.

4.1 Operational Profile

The term *profile*, as defined in [Musa96], denotes "a set of disjoint alternatives called elements, each with the probability that it will occur." The operational profile is a quantitative characterization of how a software system will be used. It is desired to have a representative operational profile for evaluating software dependability by testing, although studies [Musa94, Crespo96] showed that reliability estimates are not very sensitive to operational profile errors. The procedure for developing operational profiles is discussed in detail in [Musa86]. For use in the approach proposed in this report, the operational profile should be classified by system modes and critical operations. Two key efforts are particularly important:

- Identification of critical system modes and their occurrence probabilities
- Identification of rarely executed critical operations and their occurrence probabilities

In order to demonstrate the identification of critical system modes, we identify two broad categories of systems: (1) continuously operating real-time systems, and (2) on-line protection systems. The operational profiles differ radically for the two categories: continuous input (workload may fluctuate) and intermittent input (rare events). The first category requires high availability and can tolerate component-level failures by redundancy provisions. Computer operating systems, telephone switching and air traffic control systems all fall into this category. The second category requires successful response to emergent demands and a failed response can result in the loss of life and property. The nuclear power safety system is a typical representative of this category.

In an air traffic control center, a failure of switchover from the failed, primary air to ground communication channel to a standby channel would cause a loss of communications between air traffic controllers and all planes in the affected area, creating a potential disaster. In a nuclear plant safety system, the significant challenges are events calling for shutdown and border line events in which shutdown is not required. These events occur at rates of one to five per year. Thus, critical system modes for these two categories include:

- failure recovery mode
- challenge processing mode

A recent study of failures in the US Public Switched Telephone Network showed that in the software failure category, 70% (31 out of 44) occurred in the recovery mode [Kubn97]. This finding confirms the criticality of failure recovery mode. Typically, these critical modes are also rarely executed modes. But rarely executed operations are not limited in these modes. Based on past analyses of failure data from field software systems [Sullivant91, Lee92, Tang92, Hecht93, Voas97a, Tang98], the following operations are often rarely executed and are likely to cause problems in the operational phase:

- redundancy management
- exception handling
- initialization and calibration
- processing of data in boundary conditions or out of expected ranges

After the identification of critical modes and rarely executed operations (for simplicity, *critical operations*), the next task is to estimate occurrence probabilities for these operations. Some

of these operations are determined by the normal workload, such as initialization and calibration. Some of these operations are determined by the fault arrival rate, such as redundancy management. For example, if a process control program reinitializes itself every 100 hours and if each initialization takes five minutes, the occurrence probability for the initialization operation is 5-minutes/100-hours ≈ 0.00083. If faults that require channel switching arrive once every six months in a fault-tolerant system and if each channel switching operation takes 10 minutes, the occurrence probability for the redundancy management operation is 10-minutes/6-months ≈ 0.000038.

Sometimes it is very difficult to estimate individual occurrence probabilities for critical operations. However, it may not be very difficult to determine an upper bound that the total occurrence probability of the critical operations will not exceed. That is, we have

$$P_C < P_{upp} \qquad \textbf{(7)}$$

where P_C is the total occurrence probability for the critical operations, which is unknown, and P_{upp} is an upper bound of P_C, which can be estimated. We call P_C the *importance factor* and it is a key parameter to estimate in the proposed approach. For the NASA/JPL Deep Space Network discussed below, an estimated P_{upp} is 0.01. The estimated P_{upp} can be in place of P_C to provide an upper bound of failure rate as discussed in Section 4.2.

4.2 Quantitative Analysis of Results using Importance Sampling and the Operational Profile

As discussed in the previous subsection, the operational profile is partitioned into two sets: the *critical set* (critical and rarely executed operations) and the *regular set* which includes the other operations. Let $oc_1, oc_2, ..., oc_n$ be the disjoint operations in the critical set and $or_1, or_2, ..., or_m$ be the disjoint operations in the regular set. The occurrence probabilities for these operations are denoted by $pc_1, pc_2, ..., pc_n, pr_1, pr_2, ..., pr_m$ and satisfy

$$\sum_{i=1}^{n} pc_i + \sum_{i=1}^{m} pr_i = P_C + P_R = 1 \qquad \textbf{(8)}$$

where $P_C = \sum pc_i$ is the total occurrence probability of the operations in the critical set (*importance factor*) and $P_R = \sum pr_i$ is the total occurrence probability of the operations in the regular set. Normally,

$$P_C << P_R \qquad \textbf{(9)}$$

Assume the software system has been tested extensively (but mostly by the regular set) before the start of the stress testing. As discussed later, random (or selective) testing and fault injection are feasible techniques to exercise operations in the critical set. For example, to test nuclear safety software, various challenges can be randomly generated by emulating emergency conditions. To test the fault detection and redundancy management software, various faults can be injected. In the operational profile, these operations typically have extremely low occurrence probabilities. Based on the failure biasing technique used in importance sampling simulations [Conway87], we increase the occurrence probabilities for the operations in the critical set to $pc'_1, pc'_2, ..., pc'_n$ in proporation to their original occurrence probabilities for reducing variance on estimates:

$$pc'_i = \frac{pc_i}{P_C} \qquad \textbf{(10)}$$

Then, we have

$$\sum_{i=1}^{n} pc'_i = \frac{1}{P_C} \sum_{i=1}^{n} pc_i = 1 \qquad \textbf{(11)}$$

In a test set, a number of critical operations can be exercised. In general, for test set O_S, k critical operations, $oc_{s1}, oc_{s2}, ..., oc_{sk}$, can be tested. Since these operations are disjoint, the probability that any of these operations occurs is the sum of individual probabilities for these operations. Thus, the likelihood ratio for O_S is

$$\Lambda(O_S) = \frac{P(O_S)}{P'(O_S)} = \frac{pc_{s1}+pc_{s2}+...+pc_{sk}}{pc'_{s1}+pc'_{s1}+...+pc'_{sk}} = P_C \qquad \textbf{(12)}$$

There are two basic types of stress testing: (1) *random or selective testing* and (2) *fault or error injection*. For the first type, testing is performed continuously by sampling from the critical set. For the second type, testing is done intermittently, normally injecting one fault each time. Estimation of failure rate is only meaningful for the first type of testing (methods to estimate failure rate are discussed in Section 3.4). Let λ'_C denote the failure rate estimated from the first category of testing. By Equations (6) and (7), the failure rate for the critical set in the normal operation should be

$$\lambda_C = \Lambda(O_S)\lambda'_C = P_C\lambda'_C < P_{upp}\lambda'_C \qquad \textbf{(13)}$$

where P_{upp} has been defined in Equation (7).

Under the assumptions for this approach, failures of critical operations are the major threat to system dependability and critical operations are not frequently executed. If residual failures are not limited to critical set of operations (redundancy management, exception handling, etc.), failures of the regular set should also be counted. Since we have assumed hat the failure rate for the critical set is dominant in the late testing phase or early operational phase as supported

by the data in Table 1, the failure rate estimated from the stress testing of the critical set would be close to (at least in the same order of) the total failure rate. However, to have a more conservative assessment, the following estimate can be used:

$$\lambda = \lambda_R + \lambda_C < 2\lambda_C \quad \textbf{(14)}$$

where λ is the total failure rate and λ_R is the failure rate for the regular set. Because of the dominance of λ_C, λ is bounded by $2\lambda_C$.

Combining the above two equations, an upper bound of λ is

$$\lambda_{upp} = 2P_{upp}\lambda'_C \quad \textbf{(15)}$$

As shown in the second example discussed below, the new probabilities assigned to critical operations may not be proportional to the original probabilities as recommended in Equation (10) (a recommendation for reducing variance). The "inverted operational profile" in the example can be constructed in this way:

$$p'_i = \frac{p_{\max} - p_i}{\sum_i (p_{\max} - p_i)} \quad \textbf{(16)}$$

where p_i is the original probability for operation i, p_{max} is the maximum among p_i's, and p'_i is the new probability for operation i. In such a case, the likelihood ratio for a test set will not be the same as the importance factor. It has to be calculated using the following equation for the test set:

$$\Lambda(O_S) = \frac{P(O_S)}{P'(O_S)} = \frac{p_{s1}+p_{s2}+\ldots+p_{sk}}{p'_{s1}+p'_{s1}+\ldots+p'_{sk}} \quad \textbf{(17)}$$

For the second type of stress testing, fault/error injection, an important parameter to estimate is the fault recovery probability, C, which characterizes the fault tolerance ability of the system. A similar parameter for a safety system is the probability of successful response to an emergency demand, P_S (the criticality of this parameter is discussed in Section 4.5). Methods to estimate C or P_S are discussed in Section 4.4.

4.3 Stress Testing and Data Collection

In stress testing, it is intended to exercise rare conditions as much as possible and meanwhile, to collect necessary data from the testing for dependability analysis. Stress testing methods are for the phase represented by this tail. The possible methods are random/selective sampling testing, failure mode-based fault injection, and error data injection. The following paragraphs discuss several possible stress testing methods.

4.3.1. Random/Selective Sampling Testing

This method is to test the target software by random or selective sampling from an input data set that is rich in opportunities for creating rare conditions. That is, the data set is chosen such that it is likely to produce operations in the critical set or rarely executed set of the operational profile. Random testing has long been proposed for use in the final testing stage of software to quantify its operational reliability [Duran84]. This technique is economically feasible only where the correct results of a test can be easily identified.

The stability tests of air traffic control systems, the Advanced Automation System (AAS) [Tang95a] and the VSCS mentioned previously, are examples of random/scheduled testing. A stability test is to exercise the target system with representative input data on a large configuration continuously for a number of hours (typically 72 to 120 hours). Although the duration of the test is not very long, due to the large configuration in which multiple copies of a software task are processing different input data, the accumulative execution time for the task can be as long as thousands of hours. In these tests, the execution of some critical operations (such as the switchover of air to ground communication channels discussed above) was accelerated, but by only a limited factor (10 for the switchover operation mentioned previously).

To quantify dependability using the importance sampling method, the following data need to be collected: (1) time spent on testing the critical set, (2) number of failures occurring in each component during the testing, (3) severity of each failure, and (4) if possible, time spent on the system recovery (channel switching, reconfiguration, or restart) from each failure. These data are necessary for estimating parameters such as failure rate, recovery rate, and coverage that are required by dependability models.

4.3.2 Failure Mode Based Fault Injection

Fault injection is an effective means to test software robustness, especially for fault tolerant software. Several techniques can be used to inject a fault or error to a software program such as: (1) modify the code segment of the program (fault injection) [Iyer96], (2) modify the data segment of the program (error injection) [Iyer96], and (3) robustness benchmark testing [Siewiorek93]. The failure mode based fault injection can be implemented with any of these techniques, while the error data injection to be discussed next is limited to the last two techniques.

In most cases, it is impossible to inject all possible faults because there are numerous types of faults and numerous ways a program can exercise faults. However, the effects (manifestations) of these faults are limited and can be characterized by a number of failure modes [Barton90, Siewiorek93]. Some typical failure modes are listed below:

Stop — also referred to as "crash" or "task termination", the result of an error condition that occurs when software faults

cause the task to cease processing.

Hang — the result of an error condition that occurs when software faults cause the task to cease useful processing, yet continue regular processing and memory demands.

Delayed output — the result of an error condition that occurs when software faults cause required outputs of the task to be delivered outside of acceptable time ranges.

Invalid output — the result of an error condition that occurs when software faults cause the task to deliver erroneous outputs.

The failure mode based fault injection is to generate a failure mode by injecting faults. This approach is suitable for fault tolerant software and has been used to test the AAS air traffic control system [Dilenno21]. The approach is as follows: For each software module, all possible failure modes are first identified. Then for each identified failure mode, a number of typical faults (e.g., 5) are designed and implemented in the module to induce the failure mode. After the activation of an injected fault, the software module is monitored and the following data are collected for parameter estimation: (1) result of the failure recovery (success or non-success) which is useful for estimating the "coverage" parameter used in dependability models discussed later, and (2) time spent on the failure recovery which is useful for estimating failure recovery rate.

4.3.3. Error Data Injection

Recall the VSCS failure case discussed in Section 3.1. This event was the result of three rare conditions: a memory chip fault, hardware memory error detection function disabled, and software defects in data consistency checking. A lesson learned from this event is that data errors can be caused by hardware faults and software should prevent catastrophic failures from these errors. The Ariane disaster was also triggered by faulty data and then the lack of protection code led to the failure, except that the faulty data were generated by another faulty software module. A recent study classified the causes for software failures as three categories: erroneous input data, faulty code, or a combination of the two [Voas97b]. When the faulty code is rarely executed, the third category generates typical rare conditions as shown in the VSCS and Ariane failure events. Error injection into data is to target this issue.

One way to inject erroneous data is to change system internal state. It has been shown that many hardware faults, such as CPU, memory and network faults, can be emulated by altering memory elements [Kao93]. Thus, alternation of the data segment of the program in memory is a representative approach to injecting software data errors caused by hardware faults. Data items to which errors will be injected can be randomly selected or determined in advance. In the latter case, if critical data items have been identified, values out of expected ranges can be assigned to these data items as error injection.

Another way to inject erroneous data is to corrupt input data. This approach has been applied to a nuclear safety software system by corrupting simulated sensor signals. "In this case study, safety monitoring code was found to be faulty and not functioning properly. Fault injection easily forced the software to 'think' that a hazardous event was occurring when it was not, and vice versa. Had the safety monitoring code been working properly, fault injection would not have been able to make this happen." [Voas97b]

The *robustness benchmark testing* introduced in [Siewiorek93] can be viewed as a particular kind of error data injection. A robustness benchmark is a program designed to measure how an operating system or other commonly used software reacts to erroneous inputs. In [Siewiorek93], several primitive benchmarks were developed on the UNIX system. Each primitive benchmark targets a system functionality, such as file management and memory access, by exercising system calls with erroneous input data (e.g., open a file with a too long name) or with destructive manners (open too many files). A robust system should be able to tolerate these erroneous inputs or usages by reporting errors and avoiding system crashes. Unfortunately, none of the tested commercial systems (DEC, Sun, etc.) showed a high degree of fault tolerance.

Since error data injection can induce the same failure modes discussed above [Siewiorek93], similar to the failure mode based fault injection approach, it can furnish the estimation of the coverage and failure recovery time parameters.

4.4 Parameter Estimation

Based on the data collected from stress testing, various parameters can be estimated. These parameters will then be used in the dependability modeling and evaluation discussed in the next subsection. The most frequently used parameters can be estimated using the following methods.

Assume n failures (n could be 0) have been observed in performing test on the critical set O_S and the execution time for O_S is T'. A failure rate upper bound can then be estimated by [Kececioglu93]

$$\lambda'_{upp} = \frac{\chi^2_{1-\alpha;2n+2}}{2T'} \quad \textbf{(18)}$$

where α is the significance level ($1-\alpha$ is the confidence level) and $\chi^2_{1-\alpha;2n+2}$ determines a Chi-square probability such that $P(X_{2n+2} < \chi^2_{1-\alpha;2n+2}) = 1-\alpha$. When n is zero, i.e., no failure has been observed, the equation also applies and in this case, it is equivalent to the following estimator given in [Tang95a]:

$$\lambda'_{upp} = \frac{-\ln(\alpha)}{T'} \quad \textbf{(19)}$$

Based on methods discussed in Section 3.2, λ'_{upp} can be converted to λ_{upp}. For example, by Equation (15), a field failure rate upper bound for the target software can be estimated as:

$$\lambda_{upp} = 2P_{upp}\lambda'_{upp} \quad \textbf{(20)}$$

If the parameter of interest is the probability of successful response to a demand of emergency handling or channel switching due to a failure, i.e., the parameter C or P_S discussed in Section 3.2, the following equation can be used to estimate a lower bound [Kececioglu93]:

$$C_{low} = \frac{1}{1 + \frac{n-s+1}{s} F_{1-\alpha;2(n-s)+2;2s}} \quad \textbf{(21)}$$

where n is the number of trials, s is the number of successful trials, α is the significance level, and F represents the F statistical distribution.

If $s = n$, the estimated probability would be 1 (a 100% coverage), which is usually not representative of the likely true state because of imperfect fault detection mechanisms, defects in redundancy management or safety management software, and common mode failures. Thus, a value less than 1 is preferred in conservative dependability modeling. In such a case, a lower bound of the probability can be estimated by [Tang95a]

$$C_{low} = \alpha^{\frac{1}{n}} \quad \textbf{(22)}$$

Table 2 shows the parameter estimation process using the above equations for selected software units in the AAS air traffic control system, based on failure data collected from a stability test called Checkpoint 3 [Tang95a]. In this test, all of the six selected software units experienced failures. To show a "no-failure" case, the estimation of failure rate upper bound for the software unit BS from another stability test (Checkpoint 4) is also listed in the bottom row of the table. Since these stability tests were really not the stress tests defined in this research, λ'_{upp}, instead of λ_{upp}, should be used in the dependability model for the target system (discussed in the next subsection).

Table 2. Parameter Estimation for Software in a Real Time Distributed System for Air Traffic Control

Name of Software Unit	Operational Time (hours)	No. Of Observed Fails	No. Of Covered Failss	C_{low} (80% Conf.)	λ'_{upp} (80% Conf.)	λ_{upp} (Assuming P_{upp}=0.01)
BS	1444.8	10	10	0.85	9.45×10^{-3}	1.89×10^{-4}
CCP/ CCS	1444.8	22	20	0.82	1.86×10^{-2}	3.72×10^{-4}
CIP	1444.8	15	15	0.90	1.33×10^{-2}	2.66×10^{-4}
EFC	403.2	4	4	0.67	1.67×10^{-2}	3.34×10^{-4}
FDM	403.2	11	7	0.46	3.66×10^{-2}	7.32×10^{-4}
HFC	403.2	5	4	0.51	1.96×10^{-2}	3.92×10^{-4}
BS	3523.5	0	0	N/A	4.57×10^{-4}	9.14×10^{-6}

To demonstrate parameter estimation based on importance sampling, we hypothetically assume that these stability tests were the stress tests required by our approach and that P_{upp} was estimated as 0.01. Then, λ_{upp} would take values listed in the last (rightmost) column. Compared to λ'_{upp}, these values are one to two orders of magnitude higher. If we plug these values into the model discussed in the next subsection, the availability results (Figure 4) would be increased by similar orders of magnitude.

4.5 Dependability Modeling and Evaluation

A system can be modeled hierarchically: system level, subsystem level, and lower levels of modeling as required. This hierarchical modeling approach is recommended because (1) it reduces model complexity and (2) it facilitates identifying problem areas (by comparing results from submodels). Different levels of modeling may use different model structures, depending on the architecture and behavior of the modeled system. If failures of components are relatively independent in the modeled system, the simpler structures of reliability block diagrams and k-out-of-n models suffice. This is usually the case for the high level modeling in which subsystem failures can be considered independent. If the failure of a component may affect other components, or a reconfiguration is involved upon a component failure, Markov chains and other models (e.g., Petri net) that can account for interactions between modeled components are preferred. This is usually the case for the low level modeling or any modeling where failure correlations among components are strong. This modeling approach has been applied to both continuously operating real-time systems [Tang95a, Tang98] and on-line

protection systems [Tang97].

4.5.1 An Air Traffic Control System Example

Figure 2 shows the application software model for a Sector Suite group, a submodel in an air traffic control system model [Tang95a]. Three software units, called *operational units* (OUs), are modeled in the diagram: EFC, FDM, and HFC. All of these OUs are required for the modeled group to be functioning properly. Each OU consists of a primary process and a standby process. The notation used in the model is explained as follows:

N: Normal state in which all OUs are functioning properly
F: Failure state in which the group is unable to provide required service
x: State in which OU *x* is recovering from a failure of the primary process
x_1-x_2: State in which OUs x_1 and x_2 are recovering from a failure of their primary processes
x_1-x_2-x_3: State in which OUs x_1, x_2 and x_3 are recovering from a failure of their primary processes
λ_x: Failure rate for the primary process of OU *x*
C_x: Coverage for OU *x*
μ: Recovery rate for a process failure
μ_f: Recovery rate for an OU failure

where *x*, x_1, x_2 and x_3 represent EFC, FDM and HFC.

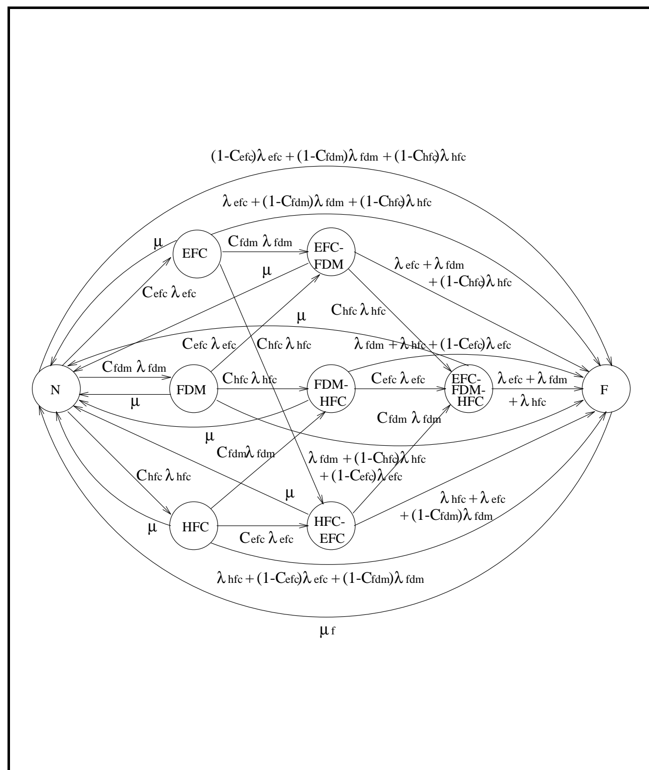

Figure 2 The Sector Suite Application Software Model

The parameters in the model were estimated from failure data collected during stability tests as shown in Table 2. The model was used to assess system availability, to identify key problem areas, and to predict required test duration for achieving desired levels of availability. Figure 3 plots the results of a prediction analysis at the 80% confidence level, where the test duration means no-failure test hours [Tang95a]. For this particular configuration (with 12 software copies running concurrently), it takes approximately 800 hours to demonstrate an availability of six 9's. If stress testing and importance sampling techniques were used as assumed in the previous subsection, failure rate λ_{upp} (table 2), instead of λ'_{upp}, would have been used in the model evaluation. Then it would be possible to achieve seven or eight 9's in the same test duration.

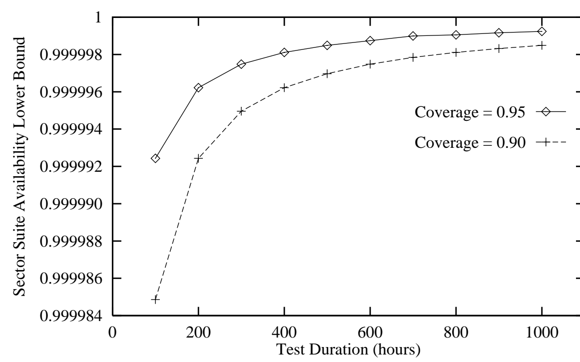

Figure 3 Availability Achieved under Various Test Durations

4.5.2. A Nuclear Power Safety System Example

The modeled configuration has two major components: a plant and a digital safety system which protects the plant by responding to and processing challenges from the plant instrumentation. A 3-level hierarchical model was developed for this configuration where levels 2 and 3 were based on the architecture of a real digital protection system — Eagle 21 [Westinghouse91]. Figure 4 shows the top-level model which reflects the intermittent operating profile discussed in Section 3.2 (the heavy frame in this diagram means parameters λss and μss are evaluated from the lower level model SafSys). The notation used in the figure is as follows:

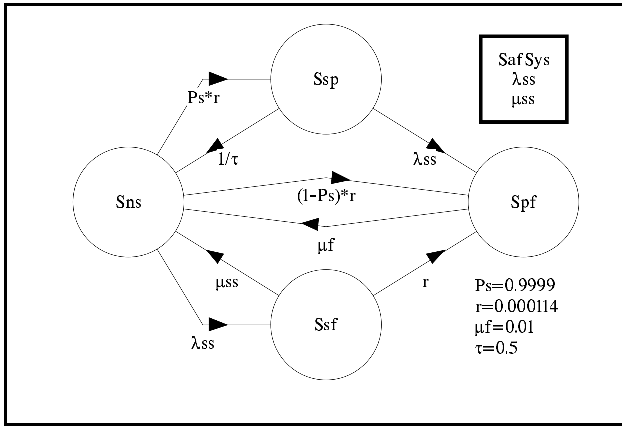

Figure 4 A Nuclear Plant Safety System Model

S_{ns} Normal/safe state in which either both plant and safety system are functioning within technical specifications or the plant is in a safe trip (reactor is shut down safely)

S_{sp} Safety processing state in which the safety system is processing a challenge

S_{sf} Safety failure state in which the safety system is not able to respond to a challenge properly while the plant is functioning with technical specifications

S_{pf} Plant failure state which is the result of a failure of the safety system to process a challenge successfully in terms of initiating a necessary reactor trip

P_S Probability of success upon demand, i.e., the safety system will be successful in responding a challenge (initially set to 0.9999)

r Arrival rate of challenges from the plant requiring a response of the safety system (assumed to be once a year, a typical value)

τ Challenge processing time (assumed to be a half hour, a conservative assumption)

λ_{ss} Failure rate of the safety system (evaluated from the lower level model)

μ_{ss} Rate for detection and handling of a safety system failure (evaluated from the lower level model)

μ_f Recovery rate of the plant after a plant failure (which has no impact on the plant MTBH)

The data source for this analysis is the failure reports generated for the early use of Eagle 21 in Unit 1 and Unit 2 at the Sequoyah plant during a 2-year period [Tang97]. The available data only allowed to estimate partial parameters required by the model. Other parameters were assumed to take typical or conservative values, as shown in the figure, for the demonstration purpose. Some key parameters were varied on reasonable ranges for analyzing their impact on dependability. The dependability measure to evaluate in this analysis is the plant Mean Time Between Hazards (MTBH), i.e., the mean time to state S_{pf} which represents a failure of the safety system to initiate a necessary reactor trip in response to a challenge due to its computer hardware or software (design or random) faults.

The results showed that P_S, the probability of success upon demand, is the most sensitive parameter, as plotted in Figure 5. When P_S increases from 0.999 to 1, MTBH increases from 1000 years to 291,000 years, i.e., an increase by 290 times. The largest increment segment is between 0.9999 and 0.99999 (from 74,500 years to 225,600 years) and achieving a value in this range is the most rewarding. It is clear that P_S is the most important parameter and achieving a high value and estimating the achieved value for this parameter should be a key effort in the system development. By Equation (22), at least 25,000 no-failure tests are required to achieve a value in this range at the 90% confidence level, assuming test cases are representative of field operations.

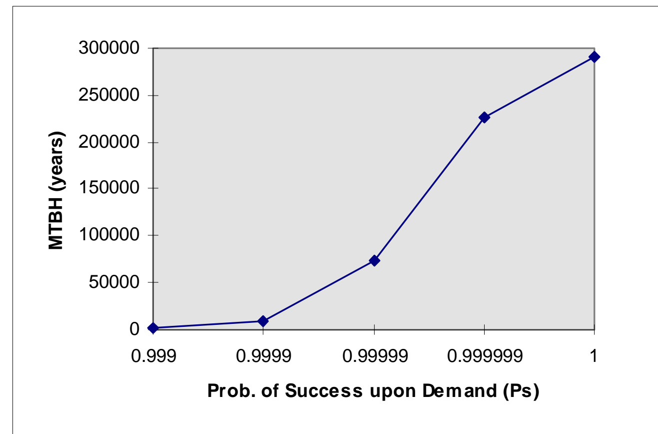

Figure 5 Sensitivity of Plant MTBH to P_S

5. Conclusion

In this paper, we have described an approach that combines the operational profile, rare conditions, importance sampling, stress testing, and measurement-based dependability evaluation in the late testing phase or early operational phase, to quantify dependability for a category of critical software constrained by the following conditions: (1) operational profile for the software is identifiable; (2) critical operations constitute only a small part of the operational profile; and (3) failures of critical operations are the major threat to system dependability. Feasible methods and possible tools to implement the approach have been identified and discussed based on real data. They include:

- Guidelines in constructing operational profiles
- Determination of the likelihood ratio
- Stress testing methods
- Parameter estimation methods
- Dependability modeling methods
- Tools and their functions required to implement the approach

Not only can the approach be used to quantify dependability for critical software systems in which rare conditions are the major threat to the system dependability, but it can also contribute significantly to enhancing the quality of the software by applying its stress testing methods to the target system. Further efforts need to be made in developing software tools to assist the implementation of the approach and in applying the approach and tools to real projects.

References

[Abdel-Ghaly86] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, Vol 12, No. 9, Sept. 1986, pp. 950-967.

[Arlat90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Transactions Software Engineering*, Vol. 16, No. 2, Feb. 1990, pp. 166-182.

[ANSI92] American National Standards Institute, *Recommended Practice for Software Reliability*, ANSI/AIAA R-03-1992, 1993.

[Avizienis85] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactions on Software Engineering*, Vol. 11, No. 12, Dec. 1985, pp. 1491-1501.

[Avresky96] D. Avresky, J. Arlat, J. C. Laprie, and Y. Crouzet, "Fault Injection for Formal Testing of Fault Tolerance," *IEEE Transactions on Reliability*, Vol. 45, No. 3, Sept. 1996, pp. 443-455.

[Butler93] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, Jan. 1993, pp. 3-12.

[Barton90] J. H. Barton, E. W. Czeck, Z. Z. Segall and D. P. Siewiorek, "Fault Injection Experiments Using FIAT," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 575-582.

[Chillarege89] R. Chillarege and N. S. Bowen, "Understanding Large System Failures — A Fault Injection Experiment," *Proceedings of the 19th International Symposium on Fault-Tolerant Computing*, June 1989, pp. 356-363.

[Choi93] G. Choi and R. K. Iyer, "Wear-Out Simulation Environment for VLSI Designs," *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, June 1993, pp. 320-329.

[Conway87] A. E. Conway and A. Goyal, "Monte Carlo Simulation of Computer System Availability/Reliability Models," *Proceedings of the 17th International Symposium on Fault-Tolerant Computing*, June 1987, pp. 230-235.

[Crespo96] A. N. Crespo, P. Matrella and A. Pasquini, "Sensitivity of Reliability Growth Models to Operational Profile Errors," *Proceedings of the Seventh International Symposium on Software Reliability Engineering*, Nov. 1996, pp. 35-44.

[Dilenno91] T. R. Dilenno, D. A. Yaskin and J. H. Barton, "Fault Tolerance Testing in the Advanced Automation System," *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, June 1991, pp. 18-25.

[Duran84] J. W. Duran and S. C. Ntafos, "An Evaluation of Random Testing," *IEEE Transactions on Software Engineering*, Vol. 10, No. 4, July 1984, pp. 438-444.

[Eckhardt91] D. E. Eckhardt, A. K. Caglayan, et al., "An Experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability," *IEEE Transactions on Software Engineering*, Vol 17, No. 7, July 1991, pp. 692-702.

[Farr96] W. Farr, "Software Reliability Modeling Survey," *Handbook of Software Reliability Engineering*, M. R. Lyu, Editor, McGraw-Hill, New York, 1996, pp. 71-117.

[Goyal92] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola, and P. W. Glynn, "A Unified Framework for Simulating Markovian Models of Highly Dependable Systems," *IEEE Transactions on Computers*, Vol. 41, No. 1, Jan. 1992, pp. 36-51.

[Gray90] J. Gray, "A Census of Tandem System Availability Between 1985 and 1990," *IEEE Transactions on Reliability*, Vol. 39, No. 4, Oct. 1990, pp. 409-418.

[Hecht93] H. Hecht, "Rare Conditions — An Important Cause of Failures," *Proceedings of the Eighth Annual Conference on Computer Assurance*, June 1993, pp. 81-85.

[Hecht94] H. Hecht and P. Crane, "Rare Conditions and Their Effect on Software Failures," *Proceedings of the Annual Reliability and Maintainability Symposium*, Jan. 1994, pp. 334-337.

[Hsueh87] M. C. Hsueh and R. K. Iyer, "A Measurement-Based Model of Software Reliability in a Production Environment," *Proceedings of the 11th Annual International Computer Software & Applications Conference*, Oct. 1987, pp. 354-360.

[Iyer96] R. K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability," *Fault-Tolerant Computer System Design*, D. K. Pradhan (ed.), Prentice Hall PTR, NJ, Feb. 1996, pp. 282-393.

[Kahn53] H. Kahn and A. W. Warshall, "Methods of Reducing Sample in Monte Carlo Computations," *Journal of the Operations Research Society of America*, Vol. 1, No. 5, 1953, pp. 263-278.

[Kanoun97] K. Kanoun, M. Kaaniche and J. C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Software*, March/April 1997, pp. 77-87.

[Kao93] W-L. Kao, R. K. Iyer, and D. Tang, "FINE: A Fault Injection and Monitor Environment for Tracing the UNIX System Behavior under Faults," *IEEE Transactions on Software Engineering*, Vol. 19, No. 11, Nov. 1993, pp. 1105-1118.

[Kececioglu93] D. Kececioglu, *Reliability and Life Testing Handbook*, Vol. 1 & 2, PTR Prentice Hall, Englewood Cliffs, NJ, 1993.

[Knight86] J. C. Knight and N. G. Leveson, "An Experimental Evaluation of the Assumptions of Independence in Multiversion Programming," *IEEE Transactions on Software Engineering*, Vol. 12, Jan. 1986, pp. 96-109.

[Laprie85] J. C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Proceedings of the 15th International Symposium on Fault-Tolerant Computing*, June 1985, pp. 2-11.

[Laprie95] J. C. Laprie, "Dependable Computing: Concepts, Limits, Challenges," *Special Issue of the 25th International Symposium on Fault-Tolerant Computing*, June 1995, pp. 42-54.

[Littlewood89] B. Littlewood, "Predicting Software Reliability," *Phil. Trans. Roy. Soc. London*, 1989, pp. 95-117.

[Lee92] I. Lee and R. K. Iyer, "Analysis of Software Halts in Tandem System," *Proceedings of the Third International Symposium on Software Reliability Engineering*, Oct. 1992, pp. 227-236.

[Lee93] I. Lee, D. Tang, R. K. Iyer and M. C. Hsueh, "Measurement-Based Evaluation of Operating System Fault Tolerance," *IEEE Transactions on Reliability*, Vol. 42, No. 2, June 1993, pp. 238-249.

[Lee95] I. Lee and R. K. Iyer, "Software Dependability in the Tandem GUARDIAN System," *IEEE Transactions on Software Engineering*, Vol. 21, No. 5, May 1995, pp. 455-467.

[Musa87] J. D. Musa, A. Iannino and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Company, 1987.

[Musa92] J. D. Musa, "The Operational Profile in Software Reliability Engineering: An Overview," *Proceedings of the Third International Symposium on Software Reliability Engineering*, Oct. 1992, pp. 140-154.

[Musa94] J. D. Musa, "Sensitivity of Field Failure Intensity to Operational Profile Errors," *Proceedings of the Fifth International Symposium on Software Reliability Engineering*, Nov. 1994, pp. 334-337.

[Musa96] J. Musa, G. Fuoco, N. Irving, D. Kropfl, and B. Juhlin, "The Operational Profile," *Handbook of Software Reliability Engineering*, M. R. Lyu, Editor, McGraw-Hill, New York, 1996 pp. 167-216.

[Prosise96] J. Prosise, *Programming Windows 95 with MFC*, Microsoft Press, 1996.

[Segall88] Z. Segall, et al., "FIAT — Fault Injection Based Automated Testing Environment," *Proceedings of the 18th International Symposium on Fault-Tolerant Computing*, June 1988, pp. 102-107.

[Siewiorek93] D. P. Siewiorek, J. J. Hudak, B. H. Suh, Z. Segall, "Development of a Benchmark to Measure System Robustness," *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, June 1993, pp. 88-97.

[Sullivant91] M. Sullivant and R. Chillarege, "Software Defects and Their Impact on System Availability — A Study of Field Failures in Operating Systems," *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, June 1991, pp. 2-9.

[Tang92] D. Tang and R. K. Iyer, "Analysis of the VAX/VMS Error Logs in Multicomputer Environments — A Case Study of Software Dependability," *Proceedings of the Third International Symposium on Software Reliability Engineering*, Oct. 1992, pp. 216-226.

[Tang95a] D. Tang and M. Hecht, "Evaluation of Software Dependability Based on Stability Test Data," *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*, June 1995, pp. 434-443.

[Tang95b] D. Tang, J. Miller and H. Hecht, *VSCS Availability Analysis Based on OT&E Stability Tests 2 and 3*, Technical Report, SoHaR Incorporated, July 1995.

[Tang96] D. Tang, M. Hecht, and H. Hecht, "A Methodology and Tool for Measurement-Based Dependability Evaluation of Digital Systems in Critical Applications," *IEEE Transactions on Nuclear Science*, Vol. 43, No. 3, June 1996, pp. 1903-1909.

[Tang97] D. Tang, M. Hecht, X. An and R. Brill, "MEADEP and Its Application in Dependability Analysis for A Nuclear Power Plant Safety System," submitted to the *Symposium on Nuclear Power Systems*, Albuquerque, New Mexico, Nov 11-14, 1997.

[Tang98] D. Tang, M. Hecht, J. Miller, L. Czekalski and J. Handal, "MEADEP and Its Applications in Evaluating Dependability for Air Traffic Control Systems," *Proceedings of the Annual Reliability and Maintainability Symposium*, Anaheim, CA, Jan. 19-22, 1998.

[Voas97a] J. Voas, "Software Fault Injection: Growing 'Safer' Systems," *Proceedings of the IEEE Aerospace Conference*, Snowmass, Co., Feb. 1997.

[Voas97b] J. Voas, G. McGraw, L. Kassab and L. Voas, "A 'Crystal Ball' for Software Liability," *IEEE Computer*, June 1997, pp. 29-36.

[Voas97c] J. Voas, F. Charron, G. McGraw, K. Miller and M. E. Friedman, "Predicting How Badly 'Good' Software Can Behave," *IEEE Software*, Vol. 14, No. 4, July/Aug. 1997, pp. 73-83.

[Westinghouse91] *EAGLE 21 Technical Description*, Westinghouse Electric Corporation, Process Control Division, Pittsburgh, PA, Jan. 1991.

[Wilson95] R. C. Wilson, *UNIX Test Tools and Benchmarks,* Prentice Hall PTR, Upper Saddle River, New Jersey, 1995.