A Statistical Assessment of Some Software Testing Strategies and Application of Experimental Design Techniques

V. N. Nair
University of Michigan, Ann Arbor, MI 48109

D. A. James
Bell Laboratories, Murray Hill, NJ 07974

W. K. Ehrlich and J. Zevallos
AT&T Laboratories, Middletown, NJ 07748

September 20, 1996

Abstract

An important problem in software testing is the efficient generation of test cases. Two classes of strategies, random and partition testing, have been discussed extensively in the software testing literature. In this paper, we provide a systematic statistical comparison of these two classes of strategies and demonstrate the usefulness of partition testing. We also show that some of the previous conclusions in the software testing literature about the inefficiency of partition testing are incorrect. The applicability of experimental design methods for partition testing is also discussed. A real application is used to illustrate the various concepts and to demonstrate the usefulness of experimental design methods for generating partitions.

Key Words: Design of experiments, partition testing, random testing, stratified sampling, test case allocation.

Running Title: Statistical Assessment of Software Testing Strategies

1 Introduction

In software testing, the input domain (the set of all possible inputs) is typically very large, and exhaustive testing can be very expensive. Therefore, there is a need for efficiently generating an appropriate set of test cases as a means to verify the correct operation of the software. Two classes of strategies, partition and random testing, have been discussed and compared in the testing literature (e.g., Duran and Wiorkowski, 1990; Duran and Ntafos, 1983; Hamlet and Taylor, 1990; Weyuker and Jeng, 1991; Miller et. al. 1992; and Tsoukalas, Duran, and Ntafos, 1993). Briefly, random testing refers to the selection of input cases according to a random sampling scheme, while partition testing refers to a family of testing strategies in which the input space is sub-divided according to some criteria. These include statement testing, data-flow testing, branch testing, path testing, and mutation testing (see Weyuker and Jeng, 1993 and the references therein).

Random and partition testing have been compared in terms of probability of failure detection, expected number of faults detected during testing, and "confidence" associated with the software. Hamlet and Taylor (1990) and Duran and Wiorkowski (1983) conclude that, for the situations they considered, partition testing is only marginally more efficient in detecting faults and that the additional set-up costs involved in partition testing may not justify its use. Weyuker and Jeng (1991) show analytically that partition testing can be good or bad depending on how the partition definition is related to the fault distribution. They indicate the need for further studies to fully explore the effectiveness of particular partition strategies being used in practice and the need for investigation of the effect of domain sizes, failure rates and test case allocation on the probability of failure detection. Hamlet and Taylor (1990) conclude that partition testing strategies need several orders of magnitude more test cases than random testing to achieve the same degree of confidence about the software.

In this paper, we provide a systematic statistical comparison of random and partition testing and re-evaluate some of the conclusions in the literature. We compare the two strategies on the basis of several criteria that have been used in software testing. Partition testing with proportional allocation is shown to perform at least as well as random testing in terms of all of these criteria. Further, when knowledge of the software unit under test (SUT) is used to develop effective partitions and allocation schemes, partition testing is seen to be considerably more efficient. Guidelines on when partition testing is likely to be useful, how the partitions should be chosen, and how the test cases should be allocated are provided. These guidelines supplement those in Weyuker and Jeng

(1991), Miller et. al. (1992), and Tsoukalas, Duran, and Ntafos (1993). We also show that the conclusions in Hamlet and Taylor (1990) about the inefficiency of partition testing are incorrect. It is shown in Section 4 that whenever partition testing is better than random testing in terms of failure detection, it is also better in inspiring confidence. Finally, the applicability of experimental design techniques for generating partitions and the use of fractional designs are discussed in Section 5. A real case study is used throughout the paper to illustrate the various concepts and to demonstrate the usefulness of experimental design techniques in software testing.

We should emphasize at the outset that our primary contributions are in formulating the statistical issues and in providing a systematic comparison of random and partition testing. As readers will see, the technical details are quite straightforward once the problems have been formulated appropriately. While some of these comparisons have been discussed in the software testing literature, they have been based on numerical calculations and simulations dealing with special cases. We provide a systematic statistical basis for the comparisons and discussion.

2 Random and Partition Testing Strategies

2.1 Random Testing

Let N be the total number of elements in the input domain, and suppose we want to *randomly* select n inputs for testing. This can be done on the basis of any probability distribution, i.e., the n inputs can be selected independently with the probability of selecting the ith unit varying according to a specified distribution. For example, the distribution can be chosen to reflect the *operational profile* which describes system usage in the field environment (Musa, 1993). Let $f(\cdot)$ be a given probability distribution that assigns probability $f(j)$ to the jth element in the input domain, $j = 1, ..., N$. Further, let $I_j = 1$ if the jth input will lead to failure and equal to zero otherwise. (We will not distinguish between different possible causes of failure for the most part. See, however, the case study in Section 2.3) The *expected failure rate* under random testing using this distribution is $\theta = \sum_{j=1}^{N} f(j) I_j$.

A special case of this is *simple random testing* where each element in the input domain is selected with equal probability. In this cases, θ reduces to D/N where D is the number of elements in the input domain that will lead to a failure.

Throughout the paper, we consider only sampling with replacement. Although sampling without-replacement is obviously more efficient, the practical implementation of a without-replacement sampling scheme is difficult and typically not cost-effective. Moreover, for the applications we are interested in, the size of the input domain and the partitions are large enough relative to the sample size so that the two are essentially equivalent. Random testing has generally been viewed as being easy to implement and therefore cost-effective. However, to actually ensure that every test case is selected independently according to a given probability distribution is not at all easy. This involves a careful definition of the input domain and the use of appropriate sampling schemes. If one tries to accomplish this indirectly, one runs risks such as incompletely covering the domain, selecting inputs from some parts of the domain more intensively, or introducing other biases.

2.2 Partition Testing

Many strategies have been discussed in the literature for partition testing (Weyuker and Jeng, 1991). For instance, path testing divides the input space into partitions whose inputs traverse the exact same software instruction path, and data flow testing combines inputs according to the executed path segments as determined by variable definitions and variable usage. Others include statement testing, branch testing, and mutation testing. Recently, statistical experimental design techniques have also been used as partition testing strategies (e.g. Brownlie, Prowse, and Phadke, 1992; Burroughs et al. 1994; Cohen, et al. 1994, Mandl, 1985).

From a statistical point of view, partition testing can be viewed as a stratified random sampling scheme. Any partition testing strategy consists of:

- a stratification of the input domain into K strata (cells) of size N_i, $i = 1, \ldots, K$;
- an allocation of n, the total number of units to be selected, into $n_i, i = 1, \ldots, K$ with $\sum_{i=1}^{K} n_i = n$.

In general, the cells may not be mutually exclusive although we will only consider such partitions in this paper. (Given any partition, it is always possible to obtain a finer partition where the cells are mutually exclusive.) We will let $\alpha_i = n_i/n$ denote the proportion of the total sample size allocated to the ith stratum, for $i = 1, \ldots, K$. Within each cell, the n_i units can be selected randomly according to some specified distribution as described under the random testing set up. We will denote by θ_i the expected failure rate in the ith stratum under the specified distribution, for $i = 1, \ldots, K$. We will also refer to the θ_i's as *partition failure rates.*

Again, we get simple random testing within each stratum as a special case of this formulation. In this case, if D_i denotes the number of failing inputs in the ith stratum, $\theta_i = D_i/N_i$, the proportion of such inputs. Let $\lambda_i = N_i/N$, the relative size of the ith stratum, for $i = 1, \ldots, K$. It is easily seen that $\sum_{i=1}^{K} \lambda_i \theta_i = \theta$ where θ is the expected failure rate under simple random testing.

It is well-known in the statistical literature that stratified sampling enjoys many advantages over simple random sampling (see, for example, Cochran, 1977). There are guidelines available that indicate when stratified sampling is likely to be useful, how the strata should be chosen, and how the sample should be allocated to the different strata. In this paper, we consider analogous results for random and partition testing and compare them on the basis of several commonly used criteria: failure detection probability, confidence assessment, expected number of failures detected, and precision of the estimators. These comparisons depend only on how the domain has been partitioned and how the test cases are allocated to the different partitions and not on the particular method used to obtain the partitions. Also, for any given partitioning of the input domain, gains in efficiency can be achieved by judiciously choosing the test allocation scheme. The importance of doing this does not seem to be fully appreciated in the software testing literature.

2.3 A Case Study

We introduce a real application at this point to illustrate the concepts. We will revisit it in later sections as we consider the various statistical issues. The software system that we were involved in was an operations support system developed at AT&T. A particular feature of this system, the *Administration Feature*, was the subject of our analysis. It dealt with the maintenance and scheduling of maintenance tasks performed by on-site work personnel. These maintenance tasks and their schedules were stored in a central database, and the Administration Feature enabled the data base information to be updated during the course of operations.

For transaction-based systems, the user interface is typically implemented through screen menus with considerable local processing of user-entered input occurring at the individual screen field level prior to the screen input being committed to the operations support system for database update. The user enters data into fields sequentially and is unable to move to a subsequent field until data entry in the current field had been validated. Consequently, for menu-driven transactions, a logical functional unit for developer testing is a single screen field. So we chose a single screen field in our Administration Feature as the SUT. An early integration test (and hence "fault prone") version of the system was frozen and placed under a separate node on the system test machine. Our analysis occurred in parallel with integration and system test. However, fault fixing and bug correcting were not performed on our version of the software.

Table 1: *Partitioning of the Input Domain Obtained by Grouping the Factors*

Factor A: Unique Tasks	Factor B: Replicates	Factor C: Replicate Type	Factor D: User Entry
A1: Single task	B1: Single	C1: Defined in Data Base	D1: Null
A2: 2-5 tasks	B2: Two or more	C2: Defined by User	D2: Invalid
A3: 6-10 tasks			D3: Partial
A4: $\geq$ 11 tasks			D4: Complete

An input domain-driven approach is a natural way to generate test cases in this application. In this approach, variables (or factors) that influence a software unit's processing are carefully identified and the settings of these variables are manipulated to generate test cases. In our case-study, knowledge about the requirements of the screen field and the software development process was used to identify four factors. The first set consisted of data base factors: A – Number of Unique Tasks; B – Replicates Per Task; and C – Replicate Type. The second set consisted of a single factor, D – the mode of user-input entry. This factor was expected to interact with the database factors with respect to the number of work items matched (i.e., single, multiple, or no matching), the type of matching (i.e., identical, partial, null) and ultimately the number of work items returned in a selection list for user selection (i.e., no selection list returned or else a selection list containing 2 or more work items being returned to the user). Consequently, the mode of data entry was expected to influence a task's database retrieval and subsequent processing.

The input domain size in this study, which is the number of all possible setting of these four factors, was 64,746. One can do random testing by selecting a sample n of these test cases using some specified distribution. We will consider only simple random testing in our comparisons. To implement partition testing, we classified the settings of the factors into groups based on knowledge of the software process. The possible values of Factor A (unique tasks) for the different work items in the data base ranged from 1 to 162. Instead of looking at all possible values, we grouped this into four categories in terms of frequency as shown in Table 1. The other factor levels were also grouped into categories as shown in Table 1. With these groupings, there were a total of $4 \times 2 \times 2 \times 4 = 64$ possible combinations with each combination corresponding to a partition of the input domain. For our partition testing scheme, we selected one test case randomly from each partition (each unit within the partition had an equal probability of selection).

We discovered two faults; one of these (Fault 2) was also detected by the system test organization while the other (Fault 1) had gone undetected by both the development and system test organizations. A root cause analysis of the two faults enabled us to understand the relationship between faults and inputs and to determine the proportion of failing inputs, θ_i's, attributable to a given fault in each of the 64 cells of the partition. Figure 1 summarizes the λ_i's and the two sets of failure rates for both faults. We will revisit this case study in subsequent sections and illustrate the various statistical issues and comparisons.

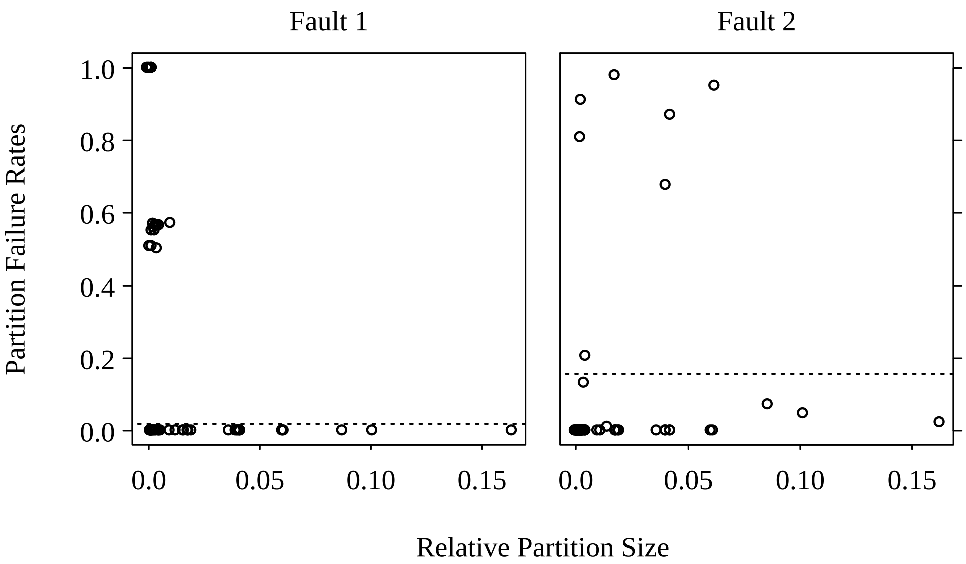

Figure 1: Detection probability θ_i versus partition size λ_i; the horizontal dotted lines depict the overall failure rate, θ, for each fault. (The points have been jittered along the horizontal direction to help distinguish overlapping symbols.)

3 Failure Detection Probability

3.1 Comparisons

An important criterion of interest in software testing is the *failure detection probability* associated with a strategy, i.e., the probability of observing at least one failure. This criterion is especially of interest when θ, the overall failure rate, is small. One then wants to maximize the probability that at least one failure is detected. In our discussion, we will consider failures due to any fault. However, as noted before, if the failures can be separated according to distinct faults (causes of failure), then we might be interested in the failure detection probabilities for each fault. For simple random sampling, the probability of detecting at least one failure based on a sample of size n is simply

$$\beta_{R,n}(\theta) = 1 - (1-\theta)^n. \tag{1}$$

For a given partition testing scheme, the corresponding failure detection probability is

$$\beta_{P,n}(\theta_1, \ldots, \theta_K) = 1 - \prod_{i=1}^{K}(1-\theta_i)^{n_i}. \tag{2}$$

Note that $\beta_{R,n}$ depends only on n and θ while $\beta_{P,n}$ depends on the sample size n, the sample allocation strategy, $n_i = n\alpha_i$'s, and the θ_i's, which are determined by the partitioning strategy. We will compare partition and random testing in terms of this criterion under different scenarios.

Let

$$\eta_P = 1 - \prod_{i=1}^{K}(1-\theta_i)^{\alpha_i}. \tag{3}$$

Then, we can re-express equation (2) as

$$\beta_{P,n}(\theta_1, \ldots, \theta_K) = 1 - (1-\eta_P)^n \equiv \beta_{R,n}(\eta_P). \tag{4}$$

This equation provides an interesting interpretation of $\beta_{P,n}$ that is extremely useful in our comparisons. *Expression (4) says that this partition testing strategy is equivalent (in the sense of having the same failure detection probability) to randomly testing a SUT that has failure rate η_P.* Thus, if $\eta_P > \theta$, this partition testing scheme will do better than random testing and if $\eta_P < \theta$, it will do worse. Thus, we can base our comparisons on just θ and η_P. This comparison has the additional advantage of being independent of the sample size n.

The following remarks are immediate from a comparison of θ and η_P.

Remark 1: If all the θ_i's are identical and hence equal θ, we have $\eta_P = \theta$ and $\beta_{R,n} = \beta_{P,n}$ for all n. Thus, if a partitioning strategy results in the θ_i's being approximately equal, then the scheme perform about the same as the random testing scheme in terms of failure detection probability. As a special case of this, if two or more cells have the same value of θ_i, then sampling separately from those cells is equivalent, in terms of this criterion, to sampling randomly from the cells combined.

Remark 2: Given a partition and hence the θ_i's, the value of η_P is maximized by the input selection scheme that allocates all the test runs to the cell with the largest value of θ_i. Similarly, it is minimized by allocating all the test runs to the cell with the smallest value of θ_i. Of course, these allocation schemes are not implementable as they depend on the unknown values of the θ_i's.

It is clear from the above that, in general, η_P can be bigger or smaller than θ and hence partition testing can be better or worse than random testing. This has led some authors to question the usefulness of partition testing. It should be intuitively clear, however, that if the way the input space is partitioned is completely unrelated to the way the failures are distributed within the input domain, then one cannot expect much advantage from partition testing. All that partitioning is doing in this case is to carve up the space randomly. In any given case, it can be better or worse, although on the average it should perform about as well as random testing. We can formalize this as follows.

Proposition 1: If the partitioning is done randomly so that the θ_i's are random variables with expected value equal to θ, then $E(\eta_P) \geq \theta$.

Proof: $E(\eta_P) = E[1 - \prod_{i=1}^{K}(1-\theta_i)^{\alpha_i}]$ which by Jensen's inequality is greater than or equal to $E[1 - \sum_{i=1}^{K} \alpha_i(1-\theta_i)] = \theta$ since $E(\theta_i) = \theta$.

While the above result assures us that, on the average, random partitioning performs as well as random testing, the more interesting question is whether it can actually do a lot better in situations where there is some natural basis for doing partitioning. In situations where the partitioning strategy is chosen carefully, one should expect gains over random testing. We will discuss efficient choices of partitions and allocation schemes in a later section. The next section shows that there exists an implementable allocation strategy that will ensure that partition testing is always at least as good as random testing.

3.2 Partition Testing with Proportional Allocation

If the goal is to just do better than random testing, then there is always an allocation scheme that, for any given partition, will perform at least as well as random testing. This is the proportional allocation scheme (denoted by PA) defined as follows.

Recall that $f(j)$ is the probability assigned to the jth unit under random testing. Let P_i denote the set of units in the ith partition, for $i = 1, \ldots, K$. Let $F_i = \sum_{j \in P_i} f(j)$, the total probability assigned to the ith cell under this distribution, and suppose that the n_i units within the ith stratum are selected randomly according to the distribution $f(j)/F_i$, for $i = 1, \ldots, K$. Then, a proportional allocation scheme corresponds to the case where the allocations $\alpha_i = F_i$, for $i = 1, \ldots, K$.

Proposition 2: Consider the proportional allocation scheme with the above set of α_i's, and let η_{PA} denote the value of η_P for this scheme. Then, $\eta_{PA} \geq \theta$ for any choice of the θ_i's. Hence $\beta_{PA,n} \geq \beta_{R,n}$ for all n.

The proof follows easily from a simple application of Jensen's inequality and from the fact that $\sum_{i=1}^{K} F_i\theta_i = \theta$. This scheme is implementable because it requires knowledge of only the cell probabilities F_i and not the θ_i's. Note that for the special case of simple random testing where $f(j) = 1/N$, we get $F_i = \lambda_i$ so that this reduces to the usual proportional to size allocation in survey sampling.

We see from the above that even in situations where partitioning is done very inefficiently, the proportional allocation scheme will ensure that partition testing is at least as good as random testing. In practice, $n_i = n\alpha_i$ have to be adjusted in order to be integers, so the inequality might be violated in some cases, but in such situations the differences between the two schemes should be small.

Remark 3: When the θ_i's are not too different (as might be the case when the partitioning strategy is not effective), the proportional allocation scheme will not be much better than random testing. This will also be true if all the θ_i's (and hence θ) are all relatively small. This follows from the fact that, for small values of θ_i's, $(1 - \eta_{PA}) = \prod_{i=1}^{K}(1 - \theta_i)^{F_i} \approx \exp(-\sum_i F_i\theta_i) = \exp(-\theta) \approx (1 - \theta)$. However, when the partitioning is informative and one or more of the θ_i's are large and quite different from the others, partition testing with proportional allocation can be quite a bit better than random testing. This will be especially true if, in addition, the overall failure rate θ is small.

3.3 Efficient Choices of Partitions and Allocation Schemes

For partition testing to be effective, knowledge of the SUT should be used in generating partitions and in allocating test cases to the different cells. The efficiency of a partition testing strategy depends more critically on the way the input domain is partitioned (i.e., the values of θ_i's) than on the allocation scheme (the α_i's). Of course, to maximize the failure detection probability, it is sufficient to have at least one value of $\theta_i = 1$. Then, $\eta_P = \beta_{P,n} = 1$ for all n. (Note that this can never be achieved by a random testing strategy unless $\theta = 1$.) Weyuker and Jeng (1991) call such a partition *revealing.* In practice, there is likely to be more than one cause of failure, and we would like to detect as many of these as possible. Thus, a more useful goal should be to make all the cells as homogeneous as possible in the sense that the θ_i's should be as close as possible to zero or one. How effectively this is accomplished can be measured through the ratio of between cells sum of squares to overall sum of squares; i.e., $\sum_{i=1}^{K} F_i(\theta_i - \theta)^2/[\theta(1-\theta)]$ (see Cochran (1977) and also Section 4.3). Another measure that provides a lower bound on the efficiency of a partition with respect to a given allocation scheme (i.e., α_i's) is $\delta_P = \sum_{i=1}^{K} \alpha_i \theta_i$. This will be discussed in more detail in Section 4 which considers comparisons in terms of the expected number of failures discovered.

We next consider comparisons of allocation schemes, i.e., the α_i's.

Proposition 3: Suppose the θ_i's are ordered so that $1 \geq \theta_1 \geq \cdots \geq \theta_K \geq 0$. Consider two allocation schemes corresponding to the vectors $\alpha^{\mathbf{A}}$ and $\alpha^{\mathbf{B}}$. If $\sum_{j=1}^{i} \alpha_j^B \geq \sum_{j=1}^{i} \alpha_j^A$ for $i = 1, \ldots, K$, then the allocation scheme $\alpha^{\mathbf{B}}$ has a higher failure detection probability than the scheme $\alpha^{\mathbf{A}}$.

This is intuitively obvious since $\alpha^{\mathbf{B}}$ allocates more weight to the larger θ_i's. The result can be proved easily. We have included a formal proof in the Appendix for the sake of completeness.

The best allocation, as noted in Remark 2, is to assign all the test cases to stratum 1 with the largest failure rate. This is not a feasible scheme as it requires complete knowledge of the θ_i's. It is possible, however, that knowledge about the software development process will suggest that some partitions are likely to have higher failure rate than others. Such information should be used in allocating the test cases. Also, as noted before, if the SUT is a new release of an existing software with enhancements, then the cells corresponding to the newer features should be tested much more intensively.

Perhaps the most commonly used allocation scheme is the constant allocation scheme where the same number of test cases are selected from each cell. (Of course, if the partitioning is completely homogeneous in the sense that the failure rate in each cell is either zero or one, then we need to select only one test unit from each cell.) In general, constant allocation will do better than proportional allocation if those cells with high failure rates are also relatively small in size. In such cases, constant allocation assigns a disproportionately large number of test cases to these cells. On the other hand, when cells with high failure rates are also relatively large, proportional allocation or random testing will do better than constant allocation.

3.4 Case Study Revisited

Let us now compare the effectiveness of random and partition testing by examining the results of the study introduced in Section 2.3.

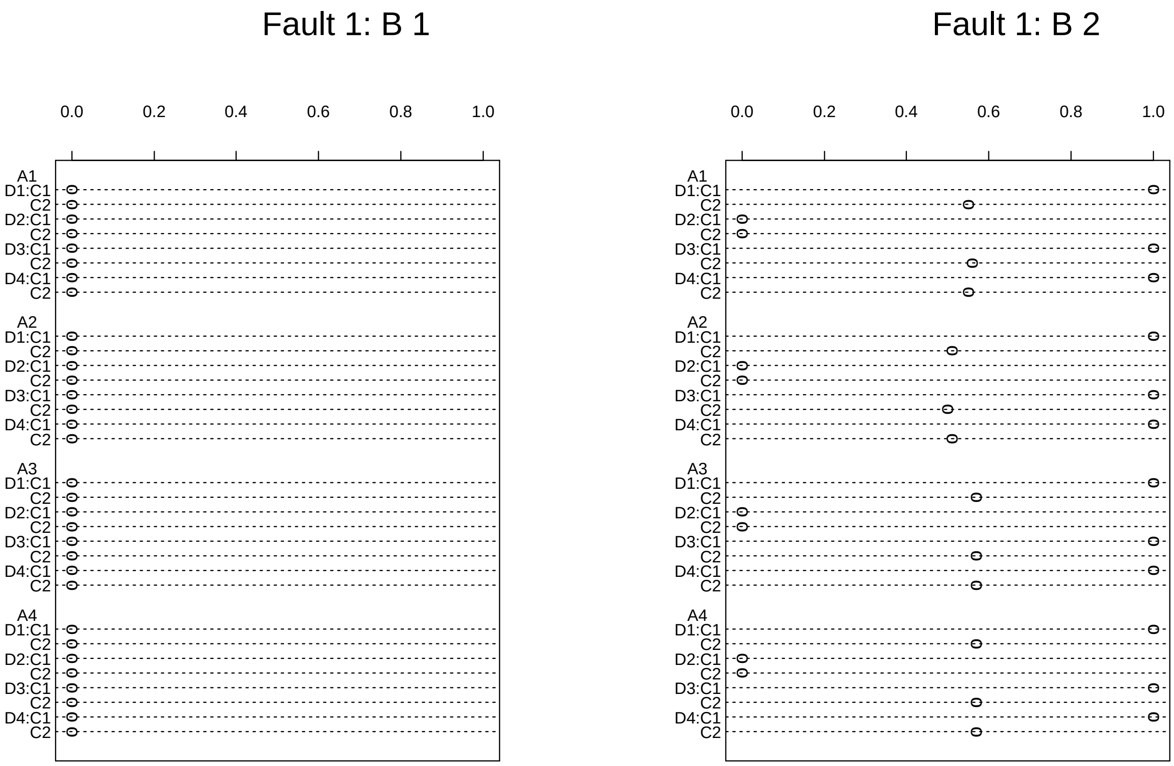

Figure 2: Distribution of detection probability across factor settings

We will analyze the two faults that were discovered separately. We will compare the partition testing strategy described in Section 2.3 with simple random testing for $n = 64$. The overall failure rate for Fault 1 was relatively small with $\theta = .02$. For this value of θ, the failure detection probability for simple random testing is $\beta_{R,64}(.02) = .74$. The corresponding failure detection probability under our partition testing scheme is one. In fact, note from Figure 1 that several of the 64 partitions had failure rate θ_i equal to one! This can be seen more clearly from Figure 2 which shows how the failure rates were distributed across the factor settings. As shown here, 12 out of the 64 partitions had failure rate equal to one, and 12 of the remaining had failure rates bigger than 0.5. Figure 1 shows that the partitions with large values of θ_i were relatively small in size (small λ_i's) thus explaining why the fault was not discovered by the development and system test groups.

Figure 2 also provides a lot of other interesting information. An examination of the different settings of Factor A shows that there is no difference in the failure rates across the different settings of Factor A. On the other hand, Factor B provides a very informative partitioning of the input domain with all of the non-zero failure rates occurring at level B2. A root-cause analysis conducted after we discovered this fault showed that this fault was due to a coding error embedded within a selection list processing statement. This error occurred only when there were multiple replicates (B2) and when the mode of user input-entry was valid (D1, D3 or D4). Among these combinations, all of the inputs with level C1 (replicate type defined in the data base) resulted in error. Because of the nature of the coding error and the data base operation, only about 50% of the inputs with level C2 (replicate type defined by user) resulted in an error. Thus, although the nature of the fault was complex, we were able to come up with a very informative partitioning of the input domain.

In fact, one could even argue that a partition test of 64 runs was inefficient in this case. As noted before, twelve out of the 64 partitions had failure rate of one, so instead of sampling from every cell, we need to select only one of these cells. In Section 5, we discuss the use of fractional factorial experimental designs for this purpose.

The situation for Fault 2 was quite different. The overall failure rate was considerably larger with $\theta = 0.16$, and simple random testing with $n = 64$ yields a failure detection probability very close to one. An examination of Figure 1 shows that our partitioning strategy did lead to some partitions with failure rates close to 1. However, their relative sizes, λ_i's, were not as small as those for Fault 1. Thus, random testing performs about as well as partition testing for this fault. Thus, not surprisingly, this fault was discovered by the development and system test groups.

4 Comparisons in Terms of Other Criteria

4.1 Confidence Assessment

Another criterion related to failure detection probability is *confidence assessment*, i.e., the "confidence" that can be attached to a SUT given that no failures were discovered during testing. This is again a criterion that is particularly useful in situations where θ, the proportion of failures in a SUT, is small. A natural measure of confidence is the upper confidence bound for the unknown value of θ based on a particular testing strategy (see, for example, Hamlet and Taylor, 1990). We will compare $U_R(\theta)$, the upper confidence bound for θ based on random testing of n elements, and $U_P(\theta)$, the corresponding bound under partition testing.

Hamlet and Taylor (1990) conclude that partition testing strategies are not useful for determining confidence associated with a software system. They claim that a partition testing strategy would require a substantially larger number of runs than random testing to achieve the same level of confidence. See their table on page 1409. On the same page, they state "Our main point has been to call into question the common wisdom that confidence in software is obtained by vigorously seeking failures and when a variety of (partition-testing) methods finds no more failures, concluding that the software will prove reliable in use. Unless the methods used employ orders of magnitude more test points, this conclusion is false. ... Clever choice of a partition in no way compensates for the disparity in these numbers." Duran and Ntafos (1980) also considered this problem and concluded that "The comparison ... shows the perhaps surprising result that random testing within program paths [i.e., partition testing] can give coarser bounds on the proportion of program failures [compared to random testing]."

Our analysis below contradicts these conclusions. We show that whenever a partition testing strategy is better than random testing for failure detection, then it is also better in inspiring confidence. Actually, this should be intuitively clear. If we use a testing strategy that is designed to have a higher probability of detecting failures and still do not detect any faults, then we should have more confidence in the software and not less.

To see this, suppose we do not detect any faults from n randomly selected tests from a SUT with failure rate μ and we want to construct an exact $(1-\alpha)$-level upper confidence bound for μ. In this situation, $U_R(\mu) = 1 - \alpha^{1/n}$ provides an exact $(1-\alpha)$-level upper confidence bound for the unknown failure rate μ (see, for example, Hahn and Meeker, 1991). Now, let us see how this can be used to compare the confidence associated with random and partition testing strategies. Consider first randomly testing n cases from a SUT with overall failure rate θ. Then, if we find no failures, we see from the above argument that $1 - \alpha^{1/n}$ is a $(1-\alpha)$-level upper confidence bound for the unknown value θ. Let us now consider a partition testing scheme with a given value of η_P (recall the definition of η_P from Section 2.1). As discussed in that section, there is an equivalence between this partition testing scheme and a random testing scheme that selects n cases randomly from a SUT with failure rate η_P. Based on this fact, if we observe no failures from partition testing, we see that $1 - \alpha^{1/n}$ is a $(1-\alpha)$-level upper bound for the unknown value η_P. Thus, whenever partition testing is better than random testing (i.e., $\theta \leq \eta_P$), we get a sharper upper bound because $1-\alpha^{1/n}$ provides an upper bound for $\eta_P \geq \theta$. Thus, we have at least as much confidence in the SUT based on partition testing as we will have using random testing. Alternatively, we require no more runs under this partition testing strategy than we require under random testing. Finally, note that there always exist a partition testing strategy that has $\eta_P \geq \theta$, namely the one with proportional allocation.

We have thus shown that a partition testing strategy that has higher probability of failure detection will also inspire higher confidence. In addition, there always exists a partition testing scheme with $\eta_P \geq \theta$, so this higher level of confidence can in fact be achieved in practice. The question remains as to why the previously cited authors arrived at the incorrect conclusions. The reason lies in the technique used to compute the necessary sample sizes (or bounds) under partition testing. It turns out that *exact* calculation of the required sample sizes for partition testing is rather complicated. What the previous authors did was to use a conservative technique which allocates the overall confidence level of α to the K partitions ($\alpha^{1/K}$, say), then computes the required test cases for each partition, and finally obtains the total number of test cases. This leads to rather conservative calculations and unduly inflates the required sample sizes under partition testing. What we have done in our analysis above is to establish an indirect comparison based on the equivalence result from the last section between partition testing and random testing. This indirect analysis is enough to establish that whenever partition testing is better than random testing in terms of failure detection probability (i.e., $\eta_P \geq \theta$), then it also provides a better confidence bound.

4.2 Expected Number of Failures Per Test Run

In situations where θ is not very small, as might be the case in early stages of software developer testing, a reasonable goal for the testing strategy is to maximize the (expected) number of failures found. Under random testing of n units, the expected number of failures is simply $n\theta$. Under partition testing, the expected number is $n\delta_P$ where $\delta_P = \sum_{i=1}^{K} \alpha_i \theta_i$. Thus, it is sufficient to compare the quantities θ and δ_P, the expected number of failures per test run which are independent of the sample size n. Recall that θ is the overall failure rate under random testing. Analogously, we can interpret δ_P as the overall failure rate under partition testing. Note that δ_P depends on both the α_i's and θ_i's.

We see that for the proportional allocation scheme where $\alpha_i = F_i, i = 1, \ldots, K$, $\delta_P = \sum_{i=1}^{K} F_i \theta_i = \theta$. Hence, for any partitioning of the input domain, however inefficient, proportional allocation again does as well as random testing. However, as discussed in Section 3, if we can exploit our knowledge about the SUT to generate efficient partitions and allocation schemes, then δ_P will be considerably larger than θ.

Proposition 4 below shows that this is true in general, i.e., δ_P provides a lower bound for η_P. So if partition testing is better than random testing in terms of the expected number of failures, then it is also better in terms of the failure detection probability. Further, we can use δ_P to obtain a lower bound on the efficiency in terms of failure detection. The proof of this proposition follows easily from an application of Jensen's inequality.

Proposition 4: $\eta_P \geq \delta_P$ with equality if and only if all the θ_i's are equal.

All of the observations made in Section 2 about efficient choices of partitioning and allocation apply to this criterion as well. We also obtain the following result that, analogous to Proposition 2, provides a comparison of two allocation schemes.

Proposition 5: Suppose the θ_i's are ordered so that $1 \geq \theta_1 \geq \cdots \geq \theta_K \geq 0$. Consider two allocation schemes corresponding to the vectors $\alpha^{\mathbf{A}}$ and $\alpha^{\mathbf{B}}$. If $\sum_{j=1}^{i} \alpha_j^B \geq \sum_{j=1}^{i} \alpha_j^A$ for $i = 1, \ldots, K$, then the expected number of failures under allocation scheme $\alpha^{\mathbf{B}}$ is larger than that under the scheme $\alpha^{\mathbf{A}}$.

This result is also intuitively obvious. A formal proof is included in the Appendix.

Remark 4: If all the θ_i's are small, then the value of δ_P will be close to η_P. Again, this can be seen by the approximations for small values of θ_i's, $\exp(-\eta_P) \approx (1-\eta_P) = \prod_i (1-\theta_i)^{\alpha_i} \approx \exp(-\sum_i \alpha_i \theta_i) = \exp(-\delta_P)$. Thus, in this case, the comparison in terms of failure detection probability is essentially the same as the comparison of expected number of failures. However, η_P can be considerably larger than δ_P if one or several of the θ_i's are larger than the others.

4.3 Precision of Estimators

Another criterion that is commonly used in the statistical literature is the precision associated with the estimator of θ under a given scheme. Let d denote the number of observed failures under random sampling. Then, $\hat{\theta}_R = d/n$ is the unbiased estimator of θ based on random testing. Similarly, under partition testing, let d_i denote the number of observed failures in the ith stratum. Then, $\hat{\theta}_i = d_i/n_i$ is an unbiased estimator of θ_i and $\hat{\theta}_P = \sum_{i=1}^{K} F_i \hat{\theta}_i$ is an unbiased estimator of θ based on the partition testing scheme. Hence we can compare the two schemes by the precision (variance) of the two unbiased estimators. If we constructed confidence intervals for θ using a normal approximation, the estimator with the smaller variance will yield narrower confidence intervals. Note that this comparison is meaningful only in situations where θ is not too small so that we actually observe failures during testing. If θ is small, the confidence assessment measure discussed earlier will be more meaningful.

We have $Var(\hat{\theta}_R) = \theta(1-\theta)/n$, and $Var(\hat{\theta}_P) = \sum_{i=1}^{K} F_i^2 \theta_i (1-\theta_i)/n\alpha_i$, which again depends on both the θ_i's and α_i's.

Proposition 6: For the proportional allocation scheme where $F_i = \alpha_i$, for $i = 1, \ldots, K$, $Var(\hat{\theta}_P) \leq Var(\hat{\theta}_R)$ with equality if and only if all the θ_i's are equal.

Proof: Under proportional allocation, $Var(\hat{\theta}_P) = \sum_{i=1}^{K} F_i \theta_i (1-\theta_i)/n = [\theta - \sum_{i=1}^{K} F_i \theta_i^2]/n$ which by Jensen's inequality is less than or equal to $[\theta - (\sum_{i=1}^{K} F_i \theta_i)^2]/n = \theta(1-\theta)/n = Var(\hat{\theta}_R)$.

So proportional allocation again performs at least as well as random testing. If partitioning and allocation are done efficiently, then partition testing can be considerably better. The efficiency of stratification (or partitioning) is usually measured in the statistical literature by the ratio of between strata sum of squares to within strata sum of squares. As noted before, the goal should be to make the between strata sum of squares as large as possible.

In terms of the allocation strategy, if the θ_i's are known, then the α_i's can be chosen to minimize $Var(\hat{\theta}_P)$. This problem has in fact been formulated even more generally in the statistical literature. Suppose there are different costs c_i's associated with the different partitions, and we want to choose the α_i's to minimize the variance subject to an overall fixed cost. (See Tsoukalas, Duran, and Ntafos (1993) for an interpretation of the c_i's in the software testing context.) Then, it can be shown (see Cochran, 1977) that the optimal choice of the α_i's is given by $alpha_i \;\; \propto \;\; F_i\{\theta_i(1-\theta_i)/c_i\}^{1/2}$. If the θ_i's are about constant, then the α_i's should be proportional to $F_i/\sqrt{c_i}$. If in addition the c_i's are constant, this reduces to proportional allocation. (See also the conclusion in Tsoukalas, Duran, and Ntafos (1993) that uses a related criterion but does worst-case comparisons.) The optimal choice depends on θ_i's which are unknown. However, as discussed before, knowledge about the software might suggest that certain cells are likely to have higher failure probabilities than the overall θ, and this information can be used to improve upon the proportional allocation scheme.

5 Application of Experimental Design Techniques

The strategy we used to partition the input domain in our case study in Section 3.2 can be viewed as an application of the experimental design methodology. We had used subject matter knowledge to identify four factors whose settings were likely to be related to the failure rates. The partitioning we obtained was a full factorial design that corresponded to the $4 \times 2 \times 2 \times 4 = 64$ combinations of these factors.

As noted in Section 3.2, 12 out of the 64 partitions had a failure rate of one for Fault 1, so a full factorial design even with just 64 test cases seems wasteful. A natural question is if we could have done just as well with a fractional design. We can investigate this issue retroactively by studying the failure detection probabilities of various fractional designs. Consider first orthogonal arrays of strength 2 where all pairwise combinations of the factors appear at least once. We need an array with run size 16 to accommodate all the factors in our study. Several possible orthogonal arrays of size 16 can be selected from the full factorial design. It turns out that all of them contain at least one of the 12 partitions with failure rate one, so all of them have a failure detection probability of one. Thus, Fault 1 could have been detected with probability one based on just 16 test cases. The corresponding probability for simple random testing with 16 runs is 0.26.

Can we do just as well with even fewer runs? Let us consider Latin hypercube designs which ensure that each level of every factor appears at least once. In our case, we need a Latin hypercube design with run size 4. There are several such designs that can be selected from the full factorial design. Not all of these designs include the partitions with failure rate one. However, the failure detection probability averaged over all such designs is 0.825. It is quite remarkable that we can achieve such a high failure detection probability by selecting only 4 test cases from a domain of size around $65,000$ with an overall failure rate of only 0.02. Note that the corresponding failure detection probability for simple random testing with run size 4 is 0.07.

The use of experimental design techniques for software testing has received attention in the software testing literature recently. In particular, applications of Latin hypercube designs and orthogonal arrays have been discussed by Mandl (1985), Sherwood (1991), Brownlie et al. (1992), and Cohen et al. (1994). As demonstrated in our application, this approach will be useful in situations where knowledge about the software development process is used to judiciously select factors whose settings are related to the failure occurrences. In practice, there may be a large number of factors, each at several levels, so the use of full factorial designs may not be practical. Fractional designs can be viewed as a sampling scheme on the space of all possible factor combinations. Sherwood (1991) and Cohen et al. (1994) discuss the use of constrained arrays that can further reduce the number of test cases. This approach generates designs which ensure that all pairwise combinations (or triplets, quadruplets, etc.) are selected at least once but they do not have the usual balance properties. There is a considerable reduction in run size in using these designs over the regular fractional factorial designs.

The ultimate goal in these approaches is to identify at least one partition with a failure rate of one. Latin hypercube designs will achieve this when all the inputs corresponding to a level of some factor will lead to failure. Similarly, the orthogonal arrays or their constrained versions will achieve it when all the inputs corresponding to a combination of some pair of factors will lead to failure. In these cases, all of the Latin hypercube designs or all of the (constrained) orthogonal arrays have the same failure detection probability of one. What happens in cases where such a perfectly revealing partition does not exist? It turns out that there is another situation where the various fractional designs of a given strength have the same failure detection probability. This situation is somewhat analogous to classical experimental design where we have additive factor effects. To be specific, suppose we have three factors A, B and C, each at $s-$levels, and let θ_{ijk} denote the failure rate at the ith level of A, jth level of B and kth level of C. Suppose we can write

$$(1-\theta_{ijk}) = (\alpha_i^A \alpha_j^B \alpha_k^C), \ i,j,k = 1,...,s, \tag{5}$$

for some values α_i^A, α_j^B, and α_k^C which depend only on the marginal settings of the individual factors A, B, and C. Under this model, any $s-$run Latin hypercube design will have the same failure detection probability given by

$$\beta_{LH} = 1 - [\prod_{i=1}^{s} \alpha_i^A \prod_{j=1}^{s} \alpha_j^B \prod_{k=1}^{s} \alpha_k^C]. \tag{6}$$

In particular, this probability equals one if any of the α's are zero which corresponds to a perfectly revealing partition. On the other hand, instead of (5), suppose we have

$$(1-\theta_{ijk}) = \alpha_{ij}^{AB} \alpha_{jk}^{BC} \alpha_{ik}^{AC} \tag{7}$$

for some values $\alpha_{ij}^{AB}, \alpha_{jk}^{BC}$ and α_{ik}^{AC} which depend only on the pairwise settings of the factors. Under the model in (7), any regular orthogonal array of strength two with run size s^2 will have the same failure detection probability given by

$$\beta_{OA} = 1 - [\prod_{i,j} \alpha_{ij}^{AB} \prod_{j,k} \alpha_{jk}^{BC} \prod_{k,l} \alpha_{i,k}^{AC}]. \tag{8}$$

Suppose now we used an orthogonal array with run size s^2 when the model in equation (5) holds. Then, $(1-\beta_{OA}) = (1-\beta_{LH})^s$. We can achieve this same failure detection probability by using a Latin hypercube design of size s and selecting s inputs randomly at each of the $s-$combinations.

These models and comparisons can be extended in an obvious manner to more general situations.

6 Conclusions

We have compared partition and random testing in terms of different criteria. Our comparisons do not take into account the costs associated with the two testing strategies. However, it should be reiterated that the simplicity and cost-effectiveness of doing random testing has been considerably overstated in practice.

We have also shown that partition testing can be quite effective in situations where knowledge about the software development process can be used to efficiently generate partitions and allocate test inputs. Experimental design techniques have great potential as a strategy for generating partitions. Further, fractional designs can be used to further subsample from the space of all possible partitions.

7 Appendix

Propositions 2 and 5 can be obtained as special cases of the following lemma.

Lemma: Let $X_1 \geq X_2 \geq \cdots \geq X_K$ and consider $\mathbf{a} = (a_1, a_2, a_3, \ldots, a_K)$ and $\mathbf{b} = (b_1, b_2, a_3, \ldots, b_K)$ such that $\sum_{j=1}^{i} b_j \geq \sum_{j=1}^{i} a_j$ for $i = 1, 2, \ldots, K-1$ and $\sum_{j=1}^{K} b_j = \sum_{j=1}^{K} a_j$. Then $\sum_{j=1}^{K} b_j X_j \geq \sum_{j=1}^{K} a_j X_j$.

Proof:

$$\begin{aligned}
\sum_{j=1}^{K} b_j X_j - \sum_{j=1}^{K} a_j X_j &= (b_1 - a_1)X_1 + (b_2 - a_2)X_2 + \cdots + (b_K - a_K)X_K \\
&\geq [(b_1 + b_2) - (a_1 + a_2)]X_2 + (b_3 - a_3)X_3 + \cdots + (b_k - a_k)X_K \\
&\geq [(b_1 + b_2 + b_3) - (a_1 + a_2 + a_3)]X_3 + (b_4 - a_4)X_4 + \cdots + (b_k - a_k)X_K \\
&\geq \vdots \\
&\geq (\sum_{j=1}^{K} b_j - \sum_{j=1}^{K} a_j)X_K \equiv 0.
\end{aligned}$$

To prove Proposition 5, apply the lemma with $X_j = \theta_j$, $a_j = \alpha_j^{\mathbf{A}}$, and $b_j = \alpha_j^{\mathbf{B}}$, $j = 1, 2, \ldots, K$. Then, it follows immediately that

$$\delta^{\mathbf{B}} \equiv \sum_{j=1}^{K} \alpha_j^{\mathbf{B}} \theta_j \geq \sum_{j=1}^{K} \alpha_j^{\mathbf{A}} \theta_j \equiv \delta^{\mathbf{A}}.$$

To prove proposition 2, we need to show that

$$\eta_P^{\mathbf{B}} = 1 - \prod_{j=1}^{K} (1 - \theta_j)^{\alpha_j^{\mathbf{B}}} \geq 1 - \prod_{j=1}^{K} (1 - \theta_j)^{\alpha_j^{\mathbf{A}}}$$

when $\theta_1 \geq \theta_2 \geq \cdots \geq \theta_K$. Let $X_j = -\log(1-\theta_j)$, $j = 1, 2, \ldots, K$. Then, we have $X_1 \geq X_2, \cdots, \geq X_K$. If we now apply the lemma with $a_j = \alpha_j^{\mathbf{A}}$ and $b_j = \alpha_j^{\mathbf{B}}$ for $j = 1, 2, \ldots, K$, we get

$$-\sum_{j=1}^{K} \alpha_j^{\mathbf{B}} \log(1-\theta_j) \geq -\sum_{j=1}^{K} \alpha_j^{\mathbf{A}} \log(1-\theta_j)$$

or

$$\prod_{j=1}^{K} (1-\theta_j)^{\alpha_j^{\mathbf{B}}} \leq \prod_{j=1}^{K} (1-\theta_j)^{\alpha_j^{\mathbf{A}}},$$

from which the result follows.

8 Acknowledgments

We thank J. D. Musa, E. J. Weyuker, and R. Viveros-Aguilera for their comments and suggestions on an earlier version of this paper. V. N. Nair's research was supported by NSF grants DMS-9304300 and DMI-9501217.

9 References

Brownlie, R., Prowse, J., and Phadke, M. S. (1992). "Robust testing of AT&T PMS/StartMail using OATS," *AT&T Technical Journal, Vol. 71, No. 3,* pp. 41-47.

Burroughs, K., Jain, A. and Erickson, T. L. (1994). "Improved Quality of Protocol Testing Through Techniques of Experimental Design," *Supercomm./ICC'94, IEEE International Conference on Communications* pp. 745-752.

Cohen, D. M., Dalal, S. R., Kajla, A. and Patton. C. (1994). "The Automatic Efficient Test Generator (AETG) System," *IEEE Transactions on Software Engineering,* pp. 303-309.

Chintapalli, A. P., Hedge, S. S., and Phadke, M. S. (1989). "Increase software/hardware test effectiveness with OATS," *Technical Memorandum,* AT&T Bell Laboratories, Murray Hill, NJ.

Cochran, W. G. (1977). "Sampling Techniques," New York: Wiley.

Duran, J. W. and Ntafos, S. C. (1983). "An evaluation of random testing," *IEEE Transactions on Software Engineering, Vol. SE-10,* pp. 438-444.

Duran, J. W. and Wiorkowski, J. J. (1980). "Quantifying software validity by sampling," *IEEE Transactions on Reliability, Vol. R-29,* pp. 141-144.

Hahn, G. J. and Meeker, W. Q. 1991. *Statistical Intervals*, Wiley: New York.

Hamlet, D. and Taylor, R. (1990). "Partition testing does not inspire confidence," *IEEE Transactions on Software engineering, Vol. 16,* pp. 1402-1411.

Mandl, R. (1980). "Orthogonal latin squares: An application of experimental design to compiler testing," *Communications of the ACM, Vol. 28, No. 10,* pp. 1054-1058.

Miller, K. W., Morrel, L. J., Noonan, R. E., Park, S. K., Nicol, D. M., Murril, B. W., and Voas, J. M. (1992). "Estimating the probability of failure when testing reveals no failures," *IEEE Transactions on Software Engineering, Vol. 18,* pp. 33-43.

Musa, J. D. (1993). "Operational profiles in software reliability testing," *IEEE Transactions on Software Engineering,* o *Vol. 10, No. 2,* pp. 14-32.

Sherwood, G. (1991). "Constrained Array Test System (CATS)," AT&T Bell Laboratories, Murray Hill, NJ, User's Manual.

Tsoukalas, M. Z., Duran, J. W., and Ntafos, S. C. (1993). "On some reliability estimation problems in random and partition testing," *IEEE Transactions on Software Engineering, Vol. 19,* pp. 687-697.

Weyuker, E. J. and Jeng, B. (1991). "Analyzing partition testing strategies," *IEEE Transactions on software Engineering, Vol. 17,* pp. 703-711.