Analyzing Partition Testing Strategies

Elaine J. Weyuker and Bingchiang Jeng

Abstract—**In this paper, partition testing strategies are assessed analytically. An investigation of what conditions affect the efficacy of partition testing is performed, and comparisons of the fault detection capabilities of partition testing and random testing are made. The effects of subdomain modifications on partition testing's ability to detect faults are also studied.**

Index Terms—**Partition testing, random testing, software testing.**

I. Partition Testing

The term "partition testing," in its broadest sense, refers to a very general family of testing strategies. The primary characteristic is that the program's input domain is divided into subsets, with the tester selecting one or more element from each subdomain. In the testing literature, it is common not to restrict the term "partition" to the formal mathematical meaning of a division into *disjoint* subsets, which together span the space being considered. Instead, testers generally use it in the more informal sense to refer to a division into (possibly overlapping) subsets of the domain. The goal of such a partitioning is to make the division in such a way that when the tester selects test cases based on the subsets, the resulting test set is a good representation of the entire domain. Ideally, the partition divides the domain into subdomains with the property that within each subdomain, either the program produces the correct answer for every element or the program produces an incorrect answer for every element. In [12], we called such a subdomain *revealing.* In [5], such a subdomain was called *homogeneous.*

If a partition's subdomains are revealing, one need only randomly select an element from each subset and run the program on that test case in order to determine program faults. Informal guidelines for creating such a partition and theoretical properties are discussed in [3], [11], [12]. In practice, it is common for the division of the input domain to be into non-disjoint subsets, and it is extremely unusual for the subdomains to be truly revealing.

Although code coverage strategies are not always viewed this way, they can be considered to be partition testing strategies. For example, *statement testing* requires that sufficient test data be selected so that every statement of the program is executed at least once. This divides the input domain into non-disjoint subdomains with each subdomain consisting of all test cases that cause a particular statement to be executed. Since a given test case will generally cause many statements to be executed, it is a member of the subdomain determined by each such statement. *Branch testing* also divides the domain into non-disjoint subdomains. *Path testing* requires that sufficient test data be selected so that every path from the program's entry statement to the program's exit statement is traversed at least once. This strategy *does* divide the input domain into disjoint classes since a given test case causes exactly one path to be traversed.

Rapps and Weyuker [9], [10] introduced a family of testing strategies, known as the data flow testing criteria, which require the exercising of path segments determined by combinations of variable definitions and variable uses. For each of these criteria, the input domain is divided so that there is a subdomain corresponding to every definition-use association required by the particular criterion. A given subdomain contains every input vector that causes the particular definition-use association to be satisfied. Again, a given input may be a member of many subdomains. Note that similar criteria were proposed in [6]–[8], and that the subdomains created for these criteria will, in general, also not be disjoint.

Mutation testing [1] can also be considered a partition testing technique. Given a program P and a test set T such that P is correct on every member of T, a set of alternative programs known as *mutants* of P must be produced. Each mutant P_i is formed by modifying a single statement of P in some predefined way. Each mutant is then run on every element of T, and T is said to be *mutation adequate* for P provided that for every inequivalent mutant P_i of P, there is a $t \in T$ such that $P_i(t) \neq P(t)$.

In this case, the subdomains can be viewed as consisting of all inputs that cause a given mutant to be distinguished from the program being tested. Depending on the mutation operators used, it is possible that some inputs do not distinguish the program from any mutants. In that case, we could define a subdomain consisting of all such inputs so that the input domain is the union of the subdomains. A mutant that is equivalent to the test program would induce no subdomain.

Finally, we mention the two extreme cases of partition testing: exhaustive testing and random testing. *Exhaustive testing* requires that every element of the input domain be explicitly tested. As a partition testing technique, therefore, exhaustive testing simply corresponds to the division of the input domain into single element subdomains. The other extreme case is random testing. In this case, the partition consists of one class, namely, the entire domain. Random testing can, therefore, be viewed as a degenerate form of partition testing. Since in some of our analysis we compare partition testing and random testing, when we refer to partition testing we will generally assume that the number of subdomains is at least two.

Manuscript received May 7, 1990; revised January 22, 1991. Recommended by W. Howden. This work was supported in part by the National Science Foundation under Grants CCR-85-01614 and CCR-89-20701, and by the Office of Naval Research under Contract N00014-85-K-0414.

The authors are with the Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10012.

IEEE Log Number 9100231.

II. Previous Results

Hamlet and Taylor [5] and Duran and Ntafos [2] experimentally compared random testing and partition testing, and both report results that at first appear surprising. In particular, Duran and Ntafos did a series of simulations.to determine empirically the probability of finding at least one input that produces the wrong output for a program, using both random and partition testing. They did similar experiments to determine the expected number of errors that a test set will discover under varying assumptions for these two testing strategies. They concluded that if doing the partitioning is expensive relative to performing the same number of random tests, then random testing is likely to be more cost effective than partition testing in terms of cost per fault found. More specifically, they found that although partition testing was generally better than random testing at finding bugs for a given number of test cases, the difference in effectiveness was relatively small.

Hamlet and Taylor considered Duran and Ntafos's results counterintuitive, performed similar experiments, and obtained similar results.

We will consider an analytical rather than empirical assessment of random testing and partition testing, and investigate the circumstances under which we can expect each strategy to excel, and when we can expect them to behave equally well. We also investigate desirable characteristics of partitioning strategies.

III. Definitions

We shall assume, in general, that we are considering program P, with domain D of size d, m points of which produce incorrect outputs. We shall call such inputs *failure-causing inputs.* All other inputs will be called *correct inputs.* We will generally assume that $d \gg m$ (i.e., that most inputs are correct inputs). Let n be the number of test cases selected, and let θ denote the *failure rate*; i.e., the probability that a failure-causing input will be selected as a test case.

Although in the ideal case random selection is done based on an operational distribution, i.e., a probability distribution of inputs that the software will actually encounter during use, in practice that information is frequently not available, particularly before the software has actually been operational for some time. In addition, for many software products, the operational distribution changes as the software matures, and it is therefore meaningless to speak of *the* operational distribution. Finally, basing random selection on an operational distribution is primarily important if there is a preference for detecting faults that are most likely to occur in practice. Since Duran and Ntafos, Hamlet and Taylor, and we are basing the comparison of random and partition testing solely on the probability of finding at least one failure-causing input using random and partition testing with no such preference for some faults over others, using a uniform distribution seems most appropriate. We will therefore generally assume that every element of D is equally likely to be selected as a test case and that $\theta = m/d$.

Duran and Ntafos assumed that random testing is done based on an operational distribution, and set $\theta = \sum_{i=1}^{k} p_i\theta_i$, where p_i is the probability that a randomly chosen test case is from subdomain D_i. Of course, by properly selecting values for p_i, it is possible to change the performance of random testing relative to partition testing. In particular, random testing is best when selection is heavy in subdomains with high failure rates, and worst when most test cases fall in subdomains with low failure rates. Hamlet and Taylor set $\theta = 1/k \sum_{i=1}^{k} \theta_i$. This amounts to assuming that all the subdomains are of equal size. In this paper, when other information is lacking, p_i will be set equal to the size of subdomain i divided by d. This yields the same results as setting $\theta = m/d$. Most of our results are not affected by this assumption of random testing using a uniform distribution.

When considering random testing, one question to address is whether test case selection will be with or without replacement. Both Duran and Ntafos and Hamlet and Taylor assume that selection is done *with* replacement since they set $P_r = 1 - (1-\theta)^n$, where P_r denotes the probability of finding at least one failure-causing input in n randomly selected tests. This is a realistic assumption since one would generally expect that if random testing is to be performed in practice, test cases would be selected from a large domain, and hence the likelihood of selecting the same test case more than once would be relatively small. In addition, one of the primary attractions of random testing is that test case selection and execution can be automated. In such a case, it would probably be less economical to check whether a given test case has already been generated than to sometimes perform the same test case more than once. We too will assume random selection is done with replacement.

When partition testing is performed, the domain is divided into k subsets, $D_1, D_2, \cdots, D_k$, of size $d_1, d_2, \cdots, d_k$, and failure rates $\theta_1, \theta_2, \cdots, \theta_k$, respectively. Let m_i denote the number of inputs in subdomain i for which the program produces an incorrect output. In partition testing, the elements of a subdomain are grouped together because it is believed that they are closely related in some essential way, and that any member of the class is as good a representative as any other. Therefore, when selecting members from a subdomain, the distribution is assumed uniform, and $\theta_i = m_i/d_i$.

Again, partition testing can be done with or without replacement. This is a somewhat more complicated issue for partition testing than for random testing since it is affected by how one decides to handle a related issue. If two subdomains have a non-null intersection, then it must be decided whether the selection of a test case which is a member of this intersection should be treated as an element of one or both of the subdomains. This is turn depends on how partition testing is actually performed.

In true partition testing, the tester is supposed to choose n_i test cases from each subdomain, D_i. The fact that some of the subdomains intersect means only that the test cases in the intersection of, say, subdomains D_i and D_j have a greater chance of being selected than points which appear only in D_i or only in D_j. It is even possible that a point in the intersection will be selected from both D_i and D_j.

In practice, testers frequently do not do true partition testing, especially at the start. Rather than selecting points from each subdomain, they first select several test cases, ignoring the subdomain divisions, run the test cases, and then see which subdomains remain "untouched" or insufficiently tested. In this type of strategy the fact of subdomain intersections is very important. If a selected test case is a member of several subdomains, then it "counts" for each of them.

For most of our results we assume that n_i test cases are *always* selected from D_i, regardless of whether some of these test cases have already been selected from other subdomains. In addition, the selection of a test case for D_i will *not* be counted as one of the test cases from a different subdomain D_j, even if it is an element of D_j.

We will also assume for partition testing that test cases are drawn with replacement so that it is possible that a given test case will be selected more than once from a single subdomain or from several subdomains. Since selection within subdomains is performed randomly, this should happen rarely except if the number of test cases selected from the subdomain is large relative to the subdomain size. Of course, if for all i, $n_i = 1$, then the issue of replacement is moot. Neither Duran and Ntafos nor Hamlet and Taylor explicitly discuss these issues. We believe they have made similar assumptions for their empirical studies.

Let P_p denote the probability of finding at least one failure-causing input using partition testing with n_i test cases chosen randomly from each D_i. Then $P_p = 1 - \pi_{i=1}^{k}(1-\theta_i)^{n_i}$. Informally, P_r and P_p assess the ability of the strategies to determine whether or not a program is correct.

We shall require that $1 \le n_i \le d_i, i = 1, \cdots, k$. In practice, most strategies set $n_1 = \cdots = n_k = 1$. It is easy to show that this restriction would not be a serious limitation theoretically, since for any partitioning from which n_i test cases are selected from subdomain D_i, there is another partitioning for which only one test case is selected per subdomain, such that P_p is the same for each partition. All that is necessary to ensure is to find a $\overline{\theta}_i$ for each i, such that $\overline{\theta}_i = 1 - (1-\theta_i)^{n_i}$. Therefore, results for the restricted model with $n_1 = \cdots = n_k = 1$ hold true for the more general variable selection model.

One might wonder under what circumstances it would be desirable to select different numbers of test cases from subdomains. If we knew that some subdomain was particularly fault-prone, as in the case when we have had problems with similar test cases in the past, then that would be a subdomain that the tester would want to stress. Another situation that would warrant particular attention to the elements of a subdomain is if the consequence of a failure for its elements were particularly grave. For example, if a subdomain were defined that consisted of those inputs that should cause some important recovery procedure to be activated, then it might be important to exercise that subdomain more than most other subdomains. Therefore we will not restrict our attention to the case for which $n_1 = \cdots = n_k = 1$. We shall discuss the implications of allowing some n_i to be greater than 1 in Section V.

When comparing random testing and partition testing, we shall always assume $n = \sum_{i=1}^{k} n_i$. That is, we are comparing how the two techniques behave with the same number of test cases. Observation 15 will examine the situation when that is not the case. We also assume that it is not the case that all inputs to P produce correct output or produce incorrect output. If all inputs did produce correct outputs, then $P_r = P_p = 0$, and if all inputs produce incorrect output, then $P_r = P_p = 1$, regardless of how the partitioning is done. In these cases, it does not matter how test cases are selected. Finally, although in practice partition testing strategies do not divide the domain into disjoint subdomains, we shall assume for our analysis that the subdomains are disjoint. This is an assumption made by both Duran and Ntafos and Hamlet and Taylor. As mentioned earlier, given that a predetermined number of test cases are always selected independently from each subdomain, the fact that the subdomains are or are not disjoint is not very important. It would only be important if the selection of a test case in one subdomain counted as satisfying a different subdomain as well. Notice, too, that for any partition testing strategy that does not divide the domain into disjoint subdomains, there is a closely related strategy that does: namely, the one in which each nonempty intersection is made a separate subdomain. This if D_i and D_j are subdomains induced by a partition testing strategy, then if D_i and D_j are not disjoint, the new strategy will have three subdomains: one consisting of all elements of D_i that are not elements of D_j, one consisting of all elements of D_j that are not elements of D_i, and one consisting of all inputs that are elements of both D_i and D_j.

IV. Designing Partitions

The first point to consider when analyzing the effectiveness of partition testing is: how are the subdomains chosen? We have described several well-known bases for dividing the domain. We will see that unless the division is done carefully, it is either a waste of time, as indicated by the Duran and Ntafos and Hamlet and Taylor experiments, or may actually have a negative effect. A good partition testing strategy groups together inputs that are likely to cause failures under the same sets of circumstances.

Our first observation identifies the optimal way of designing partitions in order to maximize P_p.

Observation 1: Assume domain D is divided into k subdomains. Then P_p is maximized if one subdomain contains only inputs that produce incorrect outputs.

Observation 1 tells us that to maximize our chances of finding a bug by partition testing, at least one subdomain should contain only inputs for which the program produces incorrect outputs. Of course, if we knew which test cases would cause failures or where the faults were in order to group the proper inputs together to maximize P_p, we would not have to perform testing. But, in fact, this observation is important in illuminating why experience tells us partition testing strategies work.

When we use a strategy like branch testing, for example, we are grouping together all inputs that cause a given branch to be executed. When the program contains bugs that cause only one or a few elements of a subdomain to produce incorrect output (such as in the case of an off-by-one error), then the density of bugs in the relevant subdomains will be relatively low, and in that case P_p will also be low. In those cases branch testing proves to be a poor strategy. If, on the other hand, many or all of the inputs traversing the branch behave incorrectly, then the failure rate for the relevant subdomains will be high, and so will P_p. Thus when we have subdomains that have a high density of failure-causing inputs, the situation is close to what we know is optimal. The same is true for most of the other well-known partition strategies discussed earlier.

In a sense, all of these partition testing strategies are fault-based, although the faults being checked for are not generally made explicit. The differences among these schemes reflect which potential faults are emphasized. When a partition testing strategy performs poorly, as many do in practice, the problem is frequently that the subdomains were created based directly on certain control flow or data flow characteristics of the program, rather than being directly fault-based. The faults that correspond to the subdomains, therefore, are frequently not likely or meaningful faults, but rather situations that are very difficult to even describe precisely. It is therefore unlikely that all, or most, of the members of a subdomain will actually be failure-causing inputs, and as we shall see, this implies that the value of P_p for that partition will be low.

Duran and Ntafos and Hamlet and Taylor found experimentally that partition testing is generally only marginally better than random testing. Observation 2 shows that not only is it possible that partition testing will only give a marginal improvement over random testing, it may actually be worse if the subdomains are poorly designed. Example 3 will demonstrate this situation.

Observation 2: Partition testing can be better, worse, or the same as random testing, depending on how the partitioning is performed.

Our first three examples show possible relationships between partition and random testing. In each of the following examples, assume the domain size is 100, 8 of which are failure-causing, and 2 test cases are to be selected. For this case, $P_r = 1 - (0.92)^2 = 0.15$. Let k denote the number of subdomains.

Example 1: Let $k = 2$, $d_1 = d_2 = 50$ and $n_1 = n_2 = 1$. Let each D_i contain four failure-causing inputs and 46 correct inputs. Then $P_p = 1 - (46/50)^2 = 0.15 = P_r$.

Our first example illustrates the case when artificial subdomain boundaries have been erected. Each of the subdomains has exactly the same failure rate as the entire domain and therefore it follows that $P_p = P_r$.

Example 2: Let $k = 2$, $d_1 = 92$, $d_2 = 8$, and all eight failure-causing inputs be in D_2. Let $n_1 = n_2 = 1$. Then $P_p = 1 > 0.15 = P_r$.

In the second example, partition testing is the perfect strategy as assessed by P_p. It is the situation described in Observation 1. Since D_2 contains only failure-causing inputs, the tester is guaranteed to select one. At first glance, this might appear to be a trivial example. It seems to say that if you know where all the faults in the program are, then you should group together all of the failure-causing inputs. But the example really illustrates why partition testing strategies work to the extent that they do.

As mentioned above, faults can be the basis of partition testing. In that case, each subdomain corresponds to a particular fault, and the elements of a subdomain are those inputs which produce the wrong output if the designated fault occurs in the program. The tester does not know *a priori* which faults are in the program, but the elements of the subdomains are grouped as if they were known. Relative to the designated set of faults, these subdomains are revealing.

Example 3: Let $k = 2$, $d_1 = 1$, $d_2 = 99$, and all eight failure-causing inputs be in D_2. Let $n_1 = n_2 = 1$. Then $P_p = 8/99 = 0.08 < 0.15 = P_r$.

Example 3 illustrates the case in which partition testing is worse than random testing. Essentially, in the partition testing case, one of the test cases has been "squandered," since it was chosen from a subdomain containing only correct inputs and D_2, the subdomain containing all of the failure-causing inputs, is almost as large as D, the entire domain. Our next observation describes precisely the worst possible performance by partition testing.

Observation 3: Assume n test cases are to be selected, and D is to be partitioned into k subdomains. Then P_p is minimized when $n_1 = \cdots = n_k = 1$, $\sum_{i=1}^{k-1} d_i = n - 1$, and $d_k = d - n + 1$, with all of the m failure-causing inputs in D_k. In this case, $P_p = m/d_k = m/(d - n + 1)$.

Observation 3 describes the worst case since for subdomain D_k, the failure rate is minimized by making its size as large as possible (namely $d-n+1$). In most cases, this partitioning will be worse than random testing. Intuitively, this is because in the random testing case, the tester gets n attempts at finding a bug, with each try having a likelihood of m/d of finding one. In this partitioning, the tester gets only one try, with almost the same failure rate, $m/(d - n + 1)$. Example 3 illustrated this worst-case behavior for the case $n = 2$. Note that it is assumed that there are at least $n-1$ correct inputs in the domain, which is consistent with our general assumption that $d \gg m$.

The next case we consider is when subdomains are of equal size and the same number of test cases are chosen from each subdomain.

Observation 4: If $d_1 = \cdots = d_k$ and $n_1 = \cdots = n_k$, then $P_p \geq P_r$. If, in addition, the failure-causing inputs are equally divided among the subdomains, so that $m_1 = \cdots = m_k$, then $P_p = P_r$.

We observe here that without knowing anything about the distribution of failure-causing inputs, if the partitioning divides the domain into equal sized subdomains, and we sample them equally, then we will never do worse than random testing. But notice that unless there is a very large number of subdomains (or the number of test cases chosen from each subdomain is large relative to its size), the assumption that $m \ll d$ means that even in the best case, when all failure-causing inputs are grouped into one subdomain, the probability of finding a failure-causing input with partition testing with equal-sized subdomains will be relatively low.

Example 1 illustrates the worst case for partition testing with equal sized subdomains that are equally sampled. For this case, $P_p = P_r$. Our next example illustrates the best case for such equal sized subdomains. Note that here $P_p > P_r$, but not by much.

Example 4: Let k, d_i, n_i be as in Example 1, and let all eight failure-causing inputs be in D_2. Then $P_p = 8/50 = 0.16 > 0.15 = P_r$.

Observation 4 is one of the few which does not hold true if we do not assume a uniform distribution for random testing. The next example shows that it is possible for $P_r > P_p$ with $d_1 = \cdots = d_k$ and $n_1 = \cdots = n_k$ if $p_i \neq d_i/d$.

Example 5: Let $d_1 = d_2 = 60$, $n_1 = n_2 = 1$, and $m_1 = 4$, $m_2 = 0$, $p_1 = 0.9$, $p_2 = 0.1$. Then $\theta = 0.9(4/60) + 0.1(0) = 0.06$. $P_r = 1-(1-0.06)^2 = 0.12 > 0.07 = 1-(1-4/60) = P_p$. Thus we have the following.

Observation 5: If $d_1 = \cdots = d_k$ and $n_1 = \cdots = n_k$ but $p_i \neq d_i/d$ for some $1 \leq i \leq k$, then partition testing can be better, worse, or the same as random testing.

To get a better feel for the effects of creating a partition with equal sized subdomains, we consider some additional examples. In the following four examples, we assume $d = 100$, $k = 10$, $d_1 = \cdots = d_{10} = 10$, and $n_1 = \cdots = n_{10} = 1$. As discussed above, P_p is maximized when all m failure-causing inputs are in one subdomain.

Example 6: Let $m = 1$. Then $P_r = 1 - (0.99)^{10} = 0.096$ and $P_p(\max) = 0.1$.

Example 7: Let $m = 2$. Then $P_r = 1 - (0.98)^{10} = 0.183$ and $P_p(\max) = 0.2$.

Example 8: Let $m = 3$. Then $P_r = 1 - (0.97)^{10} = 0.263$ and $P_p(\max) = 0.3$.

Example 9: Let $m = 4$. Then $P_r = 1 - (0.96)^{10} = 0.335$ and $P_p(\max) = 0.4$.

We see that in each case, partition testing has a higher probability of detecting at least one failure-causing input than random testing, but that the advantage of partition testing is slight. It is true that the benefit of partition testing increases as the number of inputs which produce an incorrect output increases, and hence the failure rate increases. However, even when the overall failure rate is 0.04, which Hamlet and Taylor note "is much higher than should be expected in any released program," the difference is still small. Recall, also, that these examples represent the *best case* distribution of faults for equal sized subdomains.

For a partition testing strategy to be really effective and therefore worth doing, it clearly has to perform substantially better than in these examples. It is therefore necessary that some subdomains be relatively small and contain only failure-causing inputs, or at least nearly so. This translates into requiring that subdomains focus closely on likely faults, grouping together all and only inputs which cause failures in the presence of the fault. Therefore, if the fault is present in the program, almost any test case chosen from the subdomain will result in a failure which will expose the fault, and hence P_p will be high.

We now consider the relationship between P_p and P_r if we relax the requirement that $n_1 = \cdots = n_k$, but retain the assumption that $d_1 = \cdots = d_k$. In the examples presented so far, $P_p \geq P_r$. Is it possible in this case for $P_r > P_p$?

Example 10: Assume $d = 100$, $k = 2$, $d_1 = d_2 = 50$, $n = 10$, and $m = 4$. $P_r = 1 - (0.96)^{10} = 0.34$. Assume all four failure-causing inputs are in D_1 and $n_1 = 1$, $n_2 = 9$. Then $P_p = 1 - (1 - 4/50) = 0.08 < 0.34 = P_r$.

Thus we observe the following.

Observation 6: Let D be partitioned into k subdomains of equal size. Then partition testing can be better, worse, or the same as random testing.

If, however, the failure-causing inputs are uniformly distributed (i.e., $d_1 = \cdots = d_k$ and $m_1 = \cdots = m_k$), it follows that $P_r = P_p$. In fact, it is not even necessary that all the subdomains be of equal size and the values of all of the m_i be equal. All that is necessary is that the failure rates be the same.

Observation 7: Assume that $\theta_1 = \cdots = \theta_k$. Then $\theta = \theta_i$ and $P_p = P_r$.

Since most partitioning schemes used in practice set $n_1 = \cdots = n_k = 1$, one might wonder whether that is sufficient to guarantee that $P_p \geq P_r$. Example 3 demonstrated that this is not the case. We have therefore shown the following.

Observation 8: Let D be partitioned into k subdomains and assume that $n_1 = \cdots = n_k = c$ test cases are selected from each subdomain. Then partition testing can be better, worse, or the same as random testing.

Observations 6 and 8 show that neither of the restrictions of Observation 4 by itself is sufficient to assure the superiority of partition testing.

V. Refining, Modifying, and Intersecting Subdomains

Refining Subdomains

We have seen so far that partition testing has the potential for being an excellent strategy or a poor one, depending upon how the domain subdivision has been done. It follows from Observation 1 that two subdomains are always sufficient to maximize P_p. But that is only of theoretical interest since it requires that at least one subdomain contain only failure-causing inputs. Given that we are going to select n test cases, are we better off having relatively few subdomains and selecting several test cases from each subdomain or having more subdomains and selecting only one test case from each? The answer depends on how the division of a subdomain is made. If we let P_p^O denote the value of P_p for the original (unrefined) partition, and P_p^R denote the value of P_p for the refined partition, our next observation points out that there is a way to guarantee that the refinement will never make things worse as assessed by the value of P_p.

Observation 9: Let D be partitioned into k subdomains $D_1, \cdots, D_k$. Assume $n > k$. Then there is a partition with n subdomains, constructed by dividing some of the D_i, such that P_p^R is greater than or equal to P_p^O. In particular, there is a refined partition such that $P_p^R = 1$.

Again, this result seems to be of no practical use since the way to increase P_p is by gathering together the inputs which produce an incorrect output, which in turn seems to require *a priori* knowledge of where the bugs are. But if the partition strategy naturally divides the domain according to some fault-based scheme, then one partition strategy which is more refined than another will more narrowly focus on the predetermined likely faults, and therefore be more effective than the less refined partition.

In addition, it is possible to algorithmically construct a refinement without *a priori* knowledge of the location of faults that is guaranteed not to decrease P_p. For this situation to prevail, one need only divide some subdomain into equal sized subdomains, and select an equal number of test cases from each resulting subdomain. It then follows from Observation 4 that the value of P_p for the refined partition is no worse than the original value of P_p.

This tells us that once a partition has been made, and values for the n_i's determined, there is no disadvantage, and sometimes an advantage, to further subdividing each subdomain D_i into n_i equal sized subdomains and choosing one test case from each. Thus there is never a benefit to be had from selecting more than one test case from a subdomain. Examples 6–9, however, demonstrate that this is a very conservative way of refining subdomains.

In addition, refining subdomains does not always improve the situation, as was demonstrated in Examples 1 and 3. Even when the refined subdomains are all of equal size, if the refinement gives one subdomain a high failure rate, and another subdomain a low failure rate, with most of the test cases being drawn from the subdomain with the low failure rate, this leads to a net decrease in P_p. This was illustrated in Example 10.

Observation 10: There exist partition refinements such that P_p^R is less than P_p^O.

We have looked at various cases of partition refinement. We now want to be able to analyze the net effect of performing a refinement. We observed that we can always guarantee an improvement (or at least no decrease) in the value of P_p (Observation 9), and that we have an algorithmic way of creating such a refinement (Observation 4). Observation 10, however, says that if partitioning is not done carefully, it could lead to a decreased value of P_p.

What, then, actually determines how good a partition is? We want the failure-causing inputs to be clustered within subdomains. But we have shown in Example 3 that that is not sufficient. In this example, all of the failure-causing inputs were in a single subdomain, but since the subdomain was large relative to the number of failure-causing inputs, the value for P_p was low (in fact lower than P_r.) Therefore concentrating the failure-causing inputs is not enough. What is needed is that the *density* of the failure-causing inputs within a subdomain be high.

An obvious important goal is to be able to describe precisely the circumstances under which it is advantageous to refine a subdomain. Observation 9 presented one such situation. Our next result considers the case for which we are to select two test cases from a given subdomain. How should this subdomain be divided so that the value of P_p^R is guaranteed to be better than the value of P_p^O? Observation 9 presented a conservative approach to guaranteeing that the refinement never made the situation worse than before the refinement. We next consider necessary and sufficient conditions to guarantee that the value of P_p improves as a result of the refinement.

Observation 11: Let D be a subdomain of size d for which two test cases are to be selected. Assume D is divided into two subdomains D_1 and D_2. Assume, too, that one test case is to be selected from each of these subdomains, and that $\theta_2 > \theta_1$. Then $P_p^R > P_p^O$ if and only if $d_1(d_1 - m_1) > d_2(d_2 - m_2)$.

Observation 11 now illuminates Examples 2 and 3. In Example 2, $d_1 = 92$, $d_2 = 8$, $m_1 = 0$, and $m_2 = 8$. Therefore, $\theta_2 > \theta_1$ and $d_1(d_1 - m_1) > d_2(d_2 - m_2)$, and hence $P_p^R > P_p^O$ for Example 2. In Example 3, in contrast, $d_1 = 1$, $d_2 = 99$, $m_1 = 0$, and $m_2 = 8$ so $\theta_2 > \theta_1$ but $d_1(d_1 - m_1) < d_2(d_2 - m_2)$. Therefore $P_p^R < P_p^O$ for this example.

The problem with this result is that it really does not help the tester decide when and how to refine subdomains since in order to apply this result the tester must have more information about the failure rate and fault distribution than it is realistic to expect. The next observation, while weaker than Observation 11, is somewhat more applicable.

Observation 12: Let D be a subdomain of size d for which two test cases are to be selected. Assume D is divided into two subdomains D_1 and D_2. Assume, too, that one test case is to be selected from each of these subdomains, and that $\theta_2 > \theta_1$. Then $P_p^R > P_p^O$ provided $d_1 > d_2$.

This now provides the tester with a guideline to follow. It says that if a subdomain can be divided into two subdomains in such a way that the failure rate for one of the subdomains is known to increase, then in order to insure that this refinement improves the value of P_p, that subdomain should be the smaller one. Of course, in order for this result to be useful, the tester still has to have relative information about various failure rates. This does seem possible, however, For example, if a subdomain was defined in very general terms, by focusing more closely on one or more likely faults, the tester might reasonably be able to expect to increase the failure rate for a given subdomain by subdividing it, as the next example illustrates.

Consider the following routine P whose domain is the integers in the range $[-10, 9]$:

```
input $x$
if $x \geq 0$ then $P_1$
        else $P_2$
```

where P_1 and P_2 are straightline code segments. Assume branch testing is used as the adequacy criterion. Then $D_1 = [-10, -1]$ and $D_2 = [0, 9]$.

One way to refine D_2 is to observe that a common programming fault is what is generally known as an "off-by-one error." That is, the correct predicate should have been "$x > 0$" rather than "$x \geq 0$". In that case, a reasonable refinement of D_2 into two subdomains D_{21} and D_{22} would be $D_{21} = [1, 9]$ and $D_{22} = [0]$.

Given that we know that an off-by-one error is a common fault, we would have reason to believe that $\theta_{22} > \theta_{21}$. Then since $d_{21} > d_{22}$, Observation 12 tells us that we can expect it to be a good strategy to subdivide D_2 in this way.

If this division is done, then this partition testing strategy would require that 0 be selected as a test case. If the off-by-one error is present, the fault would be guaranteed to be detected.

Let θ_1, θ_2, θ_{21}, and θ_{22} denote the failure rates for D_1, D_2, D_{21}, and D_{22}, respectively, and assume one test case is to be selected from D_1, two test cases from D_2, and one each from D_{21} and D_{22}. Then $P_p^O = 1 - (1-\theta_1)(1-\theta_2)^2$ and $P_p^R = 1 - (1-\theta_1)(1-\theta_{21})(1-\theta_{22})$.

If the off-by-one error is present, then $\theta_{22} = 1$, $P_p^R = 1$, and hence $P_p^R \geq P_p^O$.

This example illustrates the idea of dividing a subdomain so that the new subdomains more narrowly focus on particular likely faults. Notice that if the fault is not present, but some other fault causes 0 to be a failure-causing input, then again $\theta_{22} = 1$, $P_p^R = 1$, and hence $P_p^R \geq P_p^O$ as well.

If $\theta_{22} = 0$ (i.e., $\theta_{22} \neq 1$) and $\theta_{21} \neq 0$, then it follows from Observation 11 that $P_p^R > P_p^O$ iff $m_2 = 9$, i.e., $\theta_{21} = 1$. In this case $\theta_{21} > \theta_{22}$. Notice that Observation 12 does not provide information about the relationship between P_p^R and P_p^O since $d_{22} < d_{21}$.

Consider another example. Assume P, D_1, and D_2 are as above. Assume that something in the specification or the computations performed in P_1 indicates that inputs from $[8, 9]$ are more likely to contain failure-causing inputs than inputs from $[0, 7]$. This might be the case if 8 and 9 caused very large numbers to be computed, thereby causing overflow to occur. In that case we would believe that $\theta_{22} > \theta_{21}$. Then, since $d_{21} > d_{22}$, it follows from Observation 12 that we can expect that dividing D_2 in this way is a worthwhile thing to do.

Note that the converse of Observation 12 is not true as the following example demonstrates:

Example 11: Assume $d = 100$, $m = 5$, $n = 2$, $d_1 = 49$, $d_2 = 51$, $m_1 = 0$, $m_2 = 5$, and $n_1 = n_2 = 1$. Then $\theta_2 > \theta_1$ and $d_2 > d_1$, but $0.0980 = P_p^R > P_p^O = 0.0975$.

The next observation is a generalization of the previous one presented in a slightly different form.

Observation 13: Let D be a subdomain of size d for which k test cases are to be selected. Assume D is divided into k subdomains $D_1, D_2, \cdots, D_k$. Assume, too, that one test case is to be selected from each of these subdomains. Then $P_p^R > P_p^O$ provided $\sum_{i=1}^k \theta_i > k\theta$.

The above observation provides guidelines for deciding when it is beneficial to do partition testing rather than random testing, and when it is desirable to refine a subdomain. It does not, however, provide a general way of comparing different partitionings, as the next example demonstrates.

Example 12:

$$\text{Partition 1: } d_1 = 7, d_2 = 4, m_1 = 4, m_2 = 1, n_1 = n_2 = 1.$$

$$\text{Partition 2: } d_1 = 8, d_2 = 3, m_1 = 4, m_2 = 1, n_1 = n_2 = 1.$$

$$\sum_{i=1}^{k} \theta_i^1 = 0.82 < 0.83 = \sum_{i=1}^{k} \theta_i^2 \text{ but } P_p^1 = 0.679 > 0.667 = P_p^2.$$

Thus we have shown the following.

Observation 14: Consider two different ways of partitioning the same domain into k subdomains. Let θ_i^1 denote the failure rates for the subdomains of partition 1, and θ_i^2 denote the failure rates for partition 2. Let P_p^1 denote the value of P_p for partition 1, and P_p^2 denote the value for partition 2. Then $\sum_{i=1}^k \theta_i^1 > \sum_{i=1}^k \theta_i^2$ does not imply that $P_p^1 > P_p^2$.

No matter how the refinement is done, however, at some point there ceases to be a benefit from refining the subdomains. In particular, as soon as some subdomain consists solely of failure-causing inputs, further division is simply a waste of effort.

Before leaving the question of subdomain refinement, however, there is one further point to be emphasized. As we have mentioned, we always assume that we are comparing random testing and partition testing when the total number of test cases is the same. That is, we assume $n = \sum_{i=1}^k n_i$. Our next observation shows why this is important. It says simply that if we increase the number of test cases in order to do partition testing, we always improve the situation.

Observation 15: Let D be partitioned into two subdomains D_1 and D_2, and assume $n = 1$ and $n_1 = n_2 = 1$. Then $P_p > P_r$. Similarly, if the number of random test cases is increased, the value of P_r is guaranteed to increase.

Modifying Subdomains

We now consider the effect of moving elements from one subdomain to another. In particular, given subdomains D_1 and D_2 with $n_1 = n_2 = 1$, assume c elements of D_1 are moved to D_2. Assuming $d_1 > c$, and that neither of the subdomains contain only failure-causing inputs, under what circumstances does this movement improve the value of P_p? Let P_p^N denote the value of P_p for the new partition, and P_p^O denote the value of P_p for the original partition. Our next observation considers the case when the elements moved are failure-causing inputs.

Observation 16: Let D be partitioned into two subdomains D_1 and D_2, and assume $n_1 = n_2 = 1$. Consider a new partition which is exactly like the original one except that c failure-causing inputs have been moved from D_1 to D_2. Then $P_p^N > P_p^O$, provided $d_1 > d_2 + c$. If $d_1 = d_2 + c$ or if either of the subdomains contain only failure-causing inputs, then $P_p^N = P_p^O$.

Intersecting Subdomains

It is difficult to precisely analyze the effect on the value of P_p of permitting intersecting subdomains. As mentioned above, this is an important issue since most commonly used partition testing strategies generally do not have disjoint subdomains. If the tester had some sort of way of randomly selecting elements from the domain and deciding to place copies of such a point in several randomly selected subdomains, then P_p for the resulting partition would be less than P_p for the original, provided $m \ll d$. This is true because most of the replicated points would not be failure-causing inputs. Therefore the net effect of this type of subdomain intersection would be, in general, to decrease the failure rate, and hence, P_p.

Of course, in practice that is not how subdomains are intersected. When subdomains are associated (either explicitly or implicitly) with meaningful faults, a point that appears in many subdomains is one which is likely to be treated incorrectly if any of the associated faults are present in the code. If such a point is selected as a test case from any of those subdomains, therefore, even though the fault associated with *that* subdomain may not be present, the test case may nonetheless expose some (different) fault. Therefore, in practice, the fact that in partition testing strategies, subdomains are generally not disjoint has a net positive effect. In addition, if test cases are selected which are members of several subdomains and are considered as satisfying part of the requirement of test case selection from each, then either a smaller overall number of test cases will be required to satisfy the partition testing criterion, thereby reducing the cost of testing, or more testing can be performed.

VI. The Effectiveness Rate

We next address the issue of the cost/benefit trade-off for refining partitions or adding random tests. Let P_r^i denote the value of P_r when i random test cases have been performed. Then we shall call P_r^i/i the *effectiveness rate* for random testing.

Observation 17: For random testing, the effectiveness rate decreases as the number of test cases increases. That is, $P_r^i/i > P_r^{i+1}/(i+1)$ for $i \geq 1$.

The importance of this observation is that although each additional random test case increases P_r, the marginal effectiveness decreases. Therefore once P_r has reached an acceptable level, it is probably not worthwhile running additional random tests.

A somewhat different situation prevails for partition testing. Let us define the *effectiveness rate* for partition testing to be the value of P_p for the given partition, divided by the number of subdomains in the partition. Assume one of the subdomains is refined by being divided into two subdomains, yielding a partition with one more subdomain than the original. In this case, it is possible for the partition effectiveness rate to increase, even if the number of test cases selected from each subdomain is 1. Consider the following example.

Example 13: Let $d_1 = \cdots = d_{10} = 10$ and $n_1 = \cdots = n_{10} = 1$. Assume $m = 1$ and the failure-causing input is in D_1. Then $P_p = 0.1$ with effectiveness rate 0.01. Assume now that D_1 is divided into two subdomains, each of size 5. Then $P_p = 0.2$ for this new partition and the effectiveness rate is $0.2/11 = 0.018$.

VII. Comparison to Experimental Results

In [4], Hamlet summarizes the results of [5] as follows:

1) If partitions are not perfectly homogeneous, then the degree of homogeneity is not very important
2) Partition testing is improved when one or more partitions have a substantially higher probability of failure than that overall
3) Finer partitions are disadvantageous if their failure rates are uniform.

If subdomains (called partitions in [5]) are created so that they are fault-based, and the faults used as the basis of the subdomain definitions are chosen wisely, then point 1 above fails to be true. Simply stated, if a subdomain consists almost entirely of points that will be failure-causing points whenever the underlying fault is present, and that fault is likely to be present, then that subdomain will be close to revealing (or what Hamlet calls homogeneous). A partitioning will yield a high value for P_p if at least one of its subdomains has a failure rate near 1. The next example illustrates this point.

Example 14:

Partition 1: $d_1 = 10, d_2 = 500, d_3 = 490, m_1 = 8,$
$m_2 = 1, m_3 = 1, n_1 = n_2 = n_3 = 1.$
Then $P_p = 0.8.$

Partition 2: $d_1 = 300, d_2 = 300, d_3 = 400, m_1 = 3,$
$m_2 = 3, m_3 = 4, n_1 = n_2 = n_3 = 1.$
Then $P_p = 0.03.$

Example 14 demonstrates that even though the subdomains are not perfectly homogeneous in either partition 1 or partition 2, the fact that the subdomains of partition 1 are much closer to revealing (homogeneous) than those of partition 2 does make a substantial difference in the value of P_p. Furthermore, the reason that this is true is that when some subdomains are revealing or almost revealing because most of their elements are failure-causing inputs, their failure rates are much higher than the overall failure rate. (Note that the other reason a subdomain may be revealing or almost revealing is that most of its inputs are correct inputs, and hence its failure rate is usually lower than the overall failure rate.) Thus D_1 in partition 1 has a failure rate of 0.8, while the failure rates of the three subdomains in partition 2 are all 0.01. The values of P_p for the two partitionings show that partition 1 is likely to expose at least one fault, while partition 2 is not. This, of course, assumes that the fault associated with D_1 of partition 1 is in fact really likely to occur (and thus the failure rate of subdomain D_1 is really accurately presented).

Hamlet's points 1 and 2 above are actually contradictory. Point 1 states that the degree of homogeneity does not seem to matter much, while point 2 states that it does matter (since the degree of homogeneity is directly related to the failure rate). Point 3, above, is also not consistent with our findings. We showed that if the failure rates are uniform, then refining partitions (or in fact performing partitioning at all) has no effect, positive or negative.

VIII. Conclusions and Future Work

We have shown analytically that partition testing can be an excellent testing strategy or a poor one. In particular, we have shown that it depends largely on how the inputs that produce an incorrect output are concentrated within the subdomains defined by the partition. This implies that partitioning is most successful when the subdomain definitions are fault-based. Our observations help explain the Duran and Ntafos and Hamlet and Taylor empirical results and underscore the types of empirical studies needed in the future.

Although our model includes any possible technique for partition testing, it does not allow us to specifically examine the effectiveness of particular strategies. We want to know how good the partitioning schemes used in practice are. This should be determined with empirical studies.

Neither the Duran and Ntafos nor the Hamlet and Taylor studies did this. They did not consider whether or not strategies like branch testing, data flow testing, or mutation testing actually *do* divide the domain in a manner similar to the way they divided the domains. They did not address the issue of how intersecting subdomains affect the value of P_p for partitioning strategies used in practice. How does the fact that some inputs may appear in several subdomains affect a given strategy's fault detecting ability?

We want to know the effect on P_p of changing the failure rates among subdomains while keeping the domain size and total number of inputs which produce an incorrect output constant. We presented a result relating to this in Observation 16. We have described in Observation 11 necessary and sufficient conditions under which refining a partition will lead to an improvement in the probability of finding at least one failure-causing input. However, to apply this result in practice, a tester would have to know the failure rates associated with various subdomains. Since that is, in general, not known, this result does not give the practitioner a real guideline to follow. The weaker results of Observations 12 and 13 are somewhat more useful.

A very important question not yet addressed is whether or not P_p is the right way to assess partition testing. Do we really want to base this assessment solely on the probability of detecting at least one failure-causing input? Other measures of partition testing's quality should be considered.

At this point, we have some partial answers to the question: "Is partition testing worth the effort?" We know that in some cases the answer is an unqualified "yes," and in other cases the answer is "no." We have made some progress in describing precisely when each of these answers is appropriate, but, as discussed above, a substantial amount of work still needs to be done.

REFERENCES

[1] R.A. DeMillo, R.J. Lipton, and F.G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer,* vol. 11, pp. 34–41, Apr. 1978.

[2] J.W. Duran and S.C. Ntafos, "An evaluation of random testing," *IEEE Trans. Software Eng.,* vol. SE-10, pp. 438–444, July 1984.

[3] J.B. Goodenough and S.L. Gerhart, "Toward a theory of testing: Data selection criteria," in *Current Trends in Programming Methodology,* vol. 2, R.T. Yeh, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1977, pp. 44–79.

[4] R. Hamlet, "Theoretical comparison of testing methods," in *Proc. 3rd Symp. Testing, Analysis, and Verification,* Dec. 1989, pp. 28–37.

[5] R. Hamlet and R. Taylor, "Partition testing does not inspire confidence," in *Proc. 2nd Workshop on Software Testing, Verification, and Analysis,*July 1988, pp. 206–215.

[6] P.M. Herman, "A data flow analysis approach to program testing," *Australian Computer J.,* vol. 8, Nov. 1976.

[7] J.W. Laski and B. Korel, "A data flow oriented program testing strategy," *IEEE Trans. Software Eng.,* vol. SE-9, pp. 347–354, May 1983.

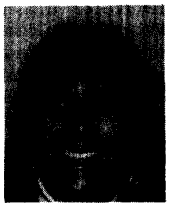
[8] S. Ntafos, "On required element testing," *IEEE Trans. Software Eng.,* vol. SE-10, pp. 795–803, Nov. 1984.

[9] S. Rapps and E.J. Weyuker, "Data flow analysis techniques for program test data selection," in *Proc. 6th Int. Conf. Software Eng.,* Sept. 1982, pp. 272–278.

[10] ——, "Selecting software test data using data flow information," *IEEE Trans. Software Eng.,* vol. SE-11, pp. 367–375, Apr. 1985.

[11] D.J. Richardson and L.A. Clarke, "A partition analysis method to increase program reliability," in *Proc. 5th Int. Conf. Software Eng.,* 1981, pp. 244–253.

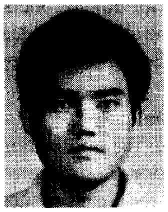
[12] E.J. Weyuker and T.J. Ostrand, "Theories of program testing and the application of revealing subdomains," *IEEE Trans. Software Eng.,* vol. SE-6, pp. 236–245, May 1980.



Elaine J. Weyuker received the M.S.E. from the Moore School of Electrical Engineering, University of Pennsylvania, and the Ph.D. in Computer Science from Rutgers University.

She is currently a professor of Computer Science at the Courant Institute of Mathematical Sciences of New York University. Before coming to NYU, she was a system's engineer for IBM and was on the faculty of the City University of New York. Her research interests are in software engineering, particularly software testing and reliability, and software complexity measures. She is also interested in the theory of computation, and is the author of (with Martin Davis) *Computability, Complexity, and Languages,* published by Academic Press.

Dr. Weyuker is secretary/treasurer of ACM SIGSOFT, and is a member of the editorial board of ACM Transactions on Software Engineering and Methodology. She is also a member of the ACM Committee on the Status of Women and Minorities, and a member of the CRA Committee on the Status of Women. She has been an ACM National Lecturer and was a member of the Executive Committee of the IEEE Computer Society Technical Committee on Software Engineering.



Bingchiang Jeng received the B.A. and M.S. degrees in computer engineering from National Chiao-Tung University in 1979 and 1981, respectively, and the Ph.D. degree in computer sciences from New York University in 1990.

He worked as a programmer for Electronic Research & Search Organization, Taiwan, in 1981. From 1984 to 1986, he was a lecturer in the Department of Electrical Engineering, National Sun Yat-Sen University, Taiwan. He is currently with the Department of Information Management, National Sun Yat-Sen University, Taiwan, as an associate professor. His current interests include software testing, software engineering, and artificial intelligence.