TESTING FOR SOFTWARE RELIABILITY

J.R. Brown and M. Lipow
TRW Systems
Redondo Beach, California 90278

Keywords and phrases. Testing, software reliability, input data space, operational profile probability distribution, representative testing, estimation of operational software reliability, input space partition, confidence, structural testing, logical paths, χ^2, program testing coverage, functional testing, expected operational usage.

Abstract. This paper presents a formulation of a novel methodology for evaluation of testing in support of operational reliability assessment and prediction. The methodology features an incremental evaluation of the representativeness of a set of development and validation test cases together with definition of additional test cases to enhance those qualities.

If test cases are derived in typical fashion (i.e., to find and remove bugs, to investigate software performance under off-nominal conditions, to exercise structural elements and functional capabilities of the software, and to demonstrate satisfaction of software requirements), then the complete set of test cases is not necessarily representative of anticipated operational usage. The paper reports on initial research into formulation of valid measures of testing representativeness.

Several techniques which permit specification of expected operational usage are described, and a technique for evaluating the correlation between actual testing accomplished and expected operational usage is defined. An unbiased estimator for operational usage reliability is proposed and justified as a function of a specified operational profile; confidence in the estimate is derived from a measure of the degree to which testing is representative of expected operational application.

An experimental application of the techniques to a small program is provided as an illustration of the proposed use of the methodology for operational software reliability estimation. The relationship between structural exercise testing thoroughness and operational usage representativeness is discussed; the specification of a quantified reliability requirement and an explicit, required representativeness measure (or confidence) is identified as integral to effective application of the proposed reliability testing methodology; efforts to extend, formalize and generalize the methodology are described; and expected benefits, as well as potential problems and limitations are identified.

The software reliability problem

The field of software quality assurance has suffered from confusion, owing to the lack of an acceptable definition of reliability and lack of means for relating quantitative measures of reliability to values that reflect actual experience with software failures. Because of the complexity of software, no adequate model of software reliability, neither conceptual nor mathematical, has been developed. A number of investigators who have studied software reliability have attempted to develop software models based on formulas and concepts borrowed from hardware reliability. These attempts have been unsuccessful, largely because the relationships between the models and actual software properties were not adequately established.

Despite the large amount of effort devoted to test and validation, undetected software errors continue to be a major concern to both designers and users. With the development of real-time software systems to control vital and critical processes, undetected errors can produce system failure with catastrophic results. Nevertheless, the goal of achieving 100 percent reliability in software by exhaustive testing is, in most cases, prohibited by cost and schedule and is, for the most part, unrealizable.

It is well known that virtually all operational programs still contain errors. In one sense then (since they are doomed to fail sooner or later), the reliability of such programs is zero (0). It is, however, equally well known that many computer programs operate day after day, and have done so for years without any errors appearing. In a different sense (i.e., based on the empirical evidence), one might be inclined to say that the reliability of such programs is one (1). This apparent paradox arises because insufficient attention has been given to the distinction between reliability in operation (or, our confidence that the program will run correctly) and reliability in theory (or, our knowledge that most programs are not completely error-free).

The main goal of this paper is to introduce a basic methodology for the evaluation of alternative software testing strategies which permits determination of the relevance of the strategies to quantitative estimation of software reliability. It is noteworthy that to date the vast majority of software reliability study and methodology development has been based upon (and thus depends upon) software testing. And yet, as formulas have been generated, data collected, and as testing-based reliability methodologies have subsequently emerged, the testing activity itself has escaped rigorous attention. Consider, for instance, the following conventional testing goals:

1) to find and remove "bugs"

2) to investigate software performance under off-nominal conditions

3) to exercise structural elements and functional capabilities of the software

4) to demonstrate satisfaction of stated software requirements.

Upon close examination we might more properly view (1) and (4) as desirable end results (i.e., goals), while (2) and (3) describe activities or testing approaches thought to be relevant to the achievement of those goals. There are empirical results, at least, which serve to support an intuitive feeling that the "means" and the "ends" are indeed related. Unfortunately, precise definition of the nature of the relationship has not been addressed, and, consequently, the meaning and value of testing has eluded the grasp of software reliability researchers. Furthermore, there is not yet a generally accepted way to begin with a statement about the relative "error-freeness" of code and produce a defensible statement about the probability that failures may (or may not) occur during operational use of the software.

If we are to achieve any real success with testing-based reliability methodologies, it is mandatory that we identify and learn to cope with significant sources of uncertainty due to insufficient knowledge of the "worth" of the testing accomplished. The remainder of this paper presents a detailed formulation and illustrated application of a particular testing-based software reliability methodology. The methodology deliberately exposes sources of uncertainty and uses a measure of the uncertainty in estimation of operational software reliability as well as determination of a quantified level of confidence in the reliability estimate. A typical application of the methodology to a particular computer program involves:

1) definition of an operational profile which characterizes expected program usage through an appropriate partition (i.e., subdivision) of the total "space" of inputs upon which the program must operate

2) specification of an operational profile probability distribution which characterizes and quantifies the relative frequency with which operational usage will expose the program to each of the subdivisions of the input space

3) for a set of test cases developed in accordance with a conventional testing strategy, a measure of the degree to which the set of tests is representative of expected operational program usage is computed

4) operational usage reliability is estimated as a function of actual testing experience factored by the operational profile probability distribution, and confidence in the reliability estimate is derived from the measure of testing representativeness, and

5) finally, the initial set of tests is augmented with additional test cases and 3 and 4 are repeated as necessary to achieve an improved reliability estimate and level of confidence.

A technical approach to reliability testing

This section presents nomenclature, definitions, assumptions and detailed formulation of fundamental elements of the above, briefly described representative testing and reliability measurement methodology. A few of the underlying concepts are not new and have been studied and documented in detail [1, 2] as part of a continuing TRW software reliability research program and related contractual efforts. These concepts (e.g., the Input Data Space) are treated here only to the extent necessary to provide a complete description of an integrated software reliability methodology.

The input data space. In a formal sense, a computer program may be defined as a specification of a computable function on the input expressions of the program. All input expressions can be represented in the form of an array of variable names $(X_1, X_2 \ldots, X_n)$, whose values need to be specified in order to cause execution of the program. The Input Data Space, E, is therefore defined as the set of $E_i \equiv (X_1, X_2, \ldots, X_n)_i$, $i = 1, 2, \ldots, N$, where each X_j can range over a finite set of values. In the general case, the number of possible values for which each X_j is defined can be very large. Consequently, the number, N, of "points" in the Input Data Space, being the product of the numbers of possible values for each X_j, could be enormous indeed.

The operational profile and its probability distribution. If we consider that any given operational use of the program is equivalent to selection of exactly one of the E_i, then the Input Data Space, as a whole, is properly viewed as a specification of an operational profile which characterizes (in fact, explicitly defines) expected operational usage. Furthermore, it is at least intuitively obvious that during operational use the program is more likely to be exposed to certain E_i than to others. In other words, if we set out to assign a "probability of selection" to each E_i, we would create an operational profile probability distribution, $P(E_i)$, which would rarely, if ever, be uniform.

Unfortunately, as indicated above, the number of E_i can be extremely large, and the effect required to create the corresponding probability distribution would be far in excess of available resources. The immediate consequence of this fact

is painfully clear. We seek a meaningful way to specify expected operational program usage, but find that the most obvious approach leads to more trouble than we can handle. Fortunately, there is a way of substantially reducing the difficulty through corresponding reduction in the number of "points" to which a probability of selection must be assigned. The reduction is effectively accomplished by viewing the Input Data Space as a collection of subspaces, that is as disjoint subsets, S_k, containing a perhaps large number of points, E_i, sharing a common characteristic. We could, for example, define S_1 as the set of all $E_i \equiv (X_1, X_2, \ldots, X_j, \ldots, X_n)_i$ such that the value of X_j is less than some specified value X, and define S_2 as the set of points for which $X_j \geq X$. To do so clearly identifies a partition of the Input Data Space, E, for which $E = S_1 \cup S_2$ and S_1 and S_2 are disjoint (i.e., $S_1 \cap S_2 = \emptyset$). If we are careful in our choice of X_j and specify a value X which has significant meaning with respect to expected operational usage, then it is possible to characterize expected usage as (for example):

$$P(S_1) = P(X_j < X) = .8 \text{ and } P(S_2) = P(X_j \geq X) = .2$$

This approach to characterizing expected operational usage involves the identification of selected intervals in the range of one or more of the input variables, X_j. The resulting partition of E into disjoint subsets, S_k, is thus defined explicitly in terms of the combinations of the disjoint intervals to which input values must belong.

There is another approach which will produce an equally useful but, in general, entirely different partition of E. To permit a ready distinction between types of partitions in later discussion, we denote this latter type as the G-partition. Subsequent reference to the Z-partition is meant to imply that the remarks apply to both the S-partition and G-partition described here. The primary principle involved in definition of the G-partition is the identification of functional characteristics of program usage in a way that allows functional groupings, G_k, of E_i to be specified. One approach to accomplishing this task involves examination of the program and identification of disjoint sets of structural elements (e.g., logic paths). In general, if we can indeed identify all of a program's structural elements and further attribute to each a statement of specific function performed, then we may rather naturally collect the structural elements into sets of elements of similar or related function. Alternatively, and preferably, we may begin with specified statements of functions to be performed by the program, and systematically define the contents of the corresponding sets by determining to which statement of function each structural element belongs.

The following discussion of logic paths is provided to illustrate the detailed attention which must be given to definition of program structural elements as necessary to support subsequent definition of the G-partition (i.e., subsets, G_k, of E) and corresponding operational profile probability distribution, $P(G_k)$.

A logic path of a program can be defined as a sequence of adjacent segments, beginning at an entry segment and proceeding by logical transfers to an exit segment. A segment is defined as follows:

1) It is a sequence of contiguous executable statements for which all statements in the segment will be executed if and only if the first statement is executed.

2) It begins with a statement to which control can be transferred and ends with a statement which transfers control to an adjacent segment.

An entry segment has no predecessor segments in the program and an exit segment results in termination or return of control to a calling program.

A logic path may be defined also in terms of the sequence of transfers of control that take place. Thus, the occurrence of the first transfer implies that an entry segment is executed followed by execution of an adjacent segment.

In general, even small programs could have a very large number of logic paths owing to the existence of loops [3]. The "large number of logic paths" problem has received a good deal of study in recent years, however, and mathematically rigorous, graph-theoretic techniques have been developed as necessary to reduce the problem to manageable proportions [4].

Let's assume that a particular program contains M logic paths of the type described above. We may label the paths, L_m, and view the total collection (i.e., $L_1, L_2, \ldots, L_M$) as a "logic path space" (L) which completely defines the program. Now, much as we did earlier with the input space, E, we are prompted to find relationships and conditions which support subdivision of L into a smaller number of disjoint sets. To do so involves identification of subspaces, L_k, containing a perhaps large number of logic paths, L_m, which share a common characteristic. We could, for example, define L_1 as the set of all L_m which contain a particular transfer (from the above definition for logic path), and define L_2 as the set of all L_m which do not contain the particular transfer. We thus define a two-way partition of the logic path space such that $L_1 \cup L_2 = L$ and $L_1 \cap L_2 = \emptyset$. There are clearly more elaborate criteria (e.g., all L_m containing two particular transfers, all L_m containing one but not both, and all L_m containing neither) which will result in a larger number of disjoint sets of L_m. In general, we need to establish the criteria in such a way that a unique function or event can be subsequently attributed to each of the subsets.

The next step is to derive a probability distribution, $P(L_k)$, which estimates the relative frequency of occurrence of the function or event associated with each of the L_k. In particular, from a knowledge of expected operational usage, one must determine the probability that an established set of criteria will be satisfied and, consequently, a program logic path from L_k will be executed.

Finally, it is important to note that a given E_i will cause exactly one logic path, L_m, to be executed. On the other hand, there may be many E_i which cause the same logic path to be executed. Thus, we may define a G-partition of E, where each G_k contains all E_i which cause execution of one of the paths in L_k. It follows from the above that the G_k are disjoint and collectively contain all of E. Moreover, note that the desired operational profile probability distribution, $P(G_k)$, is identical to the previously established $P(L_k)$.

In summary, there are at least two (and perhaps more) general approaches which permit a partition of the Input Data Space into a small number of disjoint subsets, Z_j, $j = 1, 2, \ldots, K$. Both approaches reflect prior knowledge about expected operational usage and, therefore, directly support specification of the operational profile probability distribution, $P(Z_j)$, $j = 1, \ldots, K$. Subsequently, the probability distribution can be used (as shown below) to obtain a meaningful estimate of a program's operational usage reliability as well as a measure of confidence in the estimate.

Reliability. Each input data subset, Z_j, may be further partitioned into two disjoint subsets Z_j' and Z_j'' such that

$$Z_j = Z_j' \cup Z_j''$$

Z_j' is defined as the set of points in Z_j corresponding to correct execution of the program, and Z_j'' contains the rest of the points in Z_j (i.e., those from which failures occur). Since the two subsets are disjoint, it follows that $P(Z_j') + P(Z_j'') = P(Z_j)$ and we may define the theoretical reliability, R, of a program as follows:

$$R = \sum_j P(Z_j') = 1 - \sum_j P(Z_j'')$$

Unfortunately, short of exhaustive testing, it is impossible to precisely determine the contents of Z_j' and Z_j'' or the exact values of $P(Z_j')$ and $P(Z_j'')$. At best, we are able to estimate the contents and assign probabilities based upon actual observations. For example, if a program is executed a total of n times and if f_j failures are observed out of n_j runs using points from Z_j, then an estimate of program operational usage reliability is obtained from:

$$\hat{R} = 1 - \sum_j \frac{f_j}{n_j} \cdot P(Z_j) \qquad (1)$$

The estimator, $\hat{R}$, is seen to be unbiased since the expected value of f_j is $n_j P(Z_j'')/P(Z_j)$ and, by direct substitution, the expected value of $\hat{R}$ is:

$$E(\hat{R}) = 1 - \sum_j P(Z_j'') = R$$

It is interesting to note that Equation (1) reduces to a more easily recognized and very popular estimator if we do nothing more than assume that the test cases (i.e., the n executions of the program) are identically proportional to the operational profile probability distribution. That is, if we assume

$$n_j = n \cdot P(Z_j)$$

then

$$\hat{R} = 1 - \sum_j \frac{f_j}{n_j} \cdot P(Z_j) = 1 - \sum_j \frac{f_j}{n \cdot P(Z_j)} \cdot P(Z_j)$$

so that

$$\hat{R} = 1 - \frac{1}{n} \sum_j f_j \qquad (2)$$

There is, however, very good reason to question the validity of the above assumption in the general case. In fact, some very popular testing strategies in use today regularly result in a set of test cases that is vaguely, if at all, representative of expected operational usage.

Reliability estimates derived from (1) and (2) are referred to as "the operational usage reliability" and "the observed (or assessed) reliability" and denoted as $\hat{R}_1$ and $\hat{R}_2$, respectively, in the following discussion. It is important to remember the fundamental difference between the two: that is, $\hat{R}_1$ incorporates information about expected operational program usage in order to give proper "weight" [i.e., $P(Z_j)$] to categorized success/fail experiences (i.e., f_j/n_j) observed from program testing; $\hat{R}_2$ carries with it the assumption that the accomplished testing is representative of expected operational usage and thus assumes that the individual contributions to Σf_j are properly weighted.

In either case, the reliability estimator is nothing more than a mathematical formula serving to extrapolate from a fixed number, n, of observed experiences (tests) to an unspecified number of expected experiences (operational uses of the program). When testing is, indeed, not representative of expected operational usage, we have reason to become concerned about the validity of the extrapolation. The next section introduces a technique for quantifying that concern in a way that provides a measure of confidence in the estimate of operational usage reliability.

Measuring the representativeness of testing. If we have in hand a set of n test cases and have done the work necessary to define a partition of the Input Data Space, E, and produce the operational

profile probability distribution, $P(Z_j)$, then the degree to which the testing is representative of operational usage is given by:

$$\chi^2 = \sum_{j=1}^{K} \frac{\left(n_j - nP(Z_j)\right)^2}{nP(Z_j)} \quad (3)$$

Use of the chi-square formulation yields a single, quantified measure of the extent to which the observed frequencies, n_j, are proportional to the theoretical frequencies, $n \cdot P(Z_j)$, which would (theoretically) be obtained if a random sample of n tests were drawn from the Input Data Space in accordance with $P(Z_j)$. Clearly, if actual testing is very representative of expected operational usage, then $n_j \simeq n \cdot P(Z_j)$ for j = 1, 2, ..., K, and the numerators in (3) will have values near zero. The resulting χ^2 value will therefore be very small and (entering a table of χ^2 with K-1 degrees of freedom) a very high confidence in the reliability estimate will be obtained. On the other hand, very non-representative testing will yield a large χ^2 value and, from the table of χ^2, the corresponding level of confidence may well be "significantly" low.

Finally, it is important to note that the χ^2 computation provides both 1) a measure of our confidence in the operational usage reliability estimate, and 2) information which directly supports the design of supplementary test cases as necessary to decrease the differences, $n_j - n \cdot P(Z_j)$. If properly designed, the additional test cases have the net effect of driving the χ^2 value closer to zero and yielding increased confidence in subsequently computed operational usage reliability estimates.

Representative testing and reliability measurement – an application

The preceding discussion has presented ample background information and sufficiently definitive treatment of fundamentals to support a general understanding of the primary elements of a representative testing and reliability measurement methodology. In order to more clearly depict how the methodology might be employed, a sample application is presented below. The selection of the sample program was guided, in part, by a substantial amount of previously accomplished structural testing analysis [5]. The program corresponds to a problem which Fred Gruenberger [6] stated pretty much as follows: "Determine whether three integers representing three lengths constitute an equilateral, isosceles, or scalene triangle or cannot be the sides of any triangle. Reference 5 provides a detailed description of a Triangle Type Determination Program (TTDP) and presents rationale for and results from application of certain structural-exercise test effectiveness measurement tools belonging to TRW's Product Assurance Confidence Evaluator (PACE) system.

Figure 1 presents a detailed flow diagram of TTDP, and additional descriptive information and structural analysis results are given here in Figures 2 through 5.

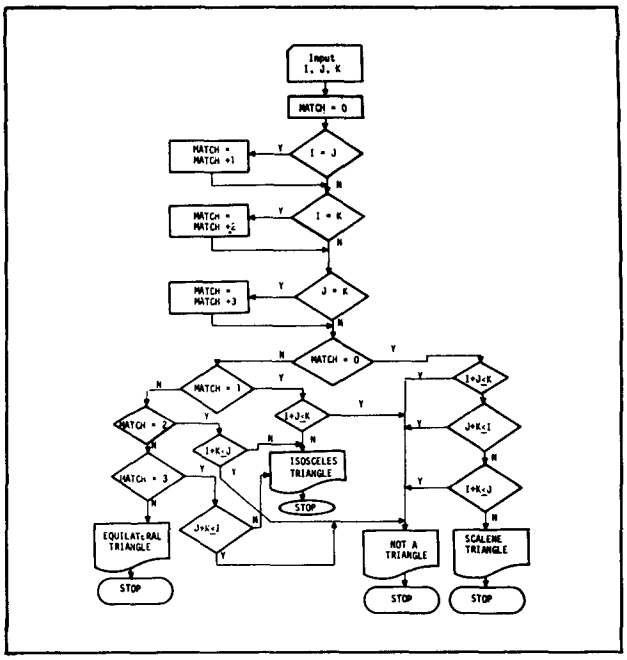

Figure 1. Flow Diagram of TTDP

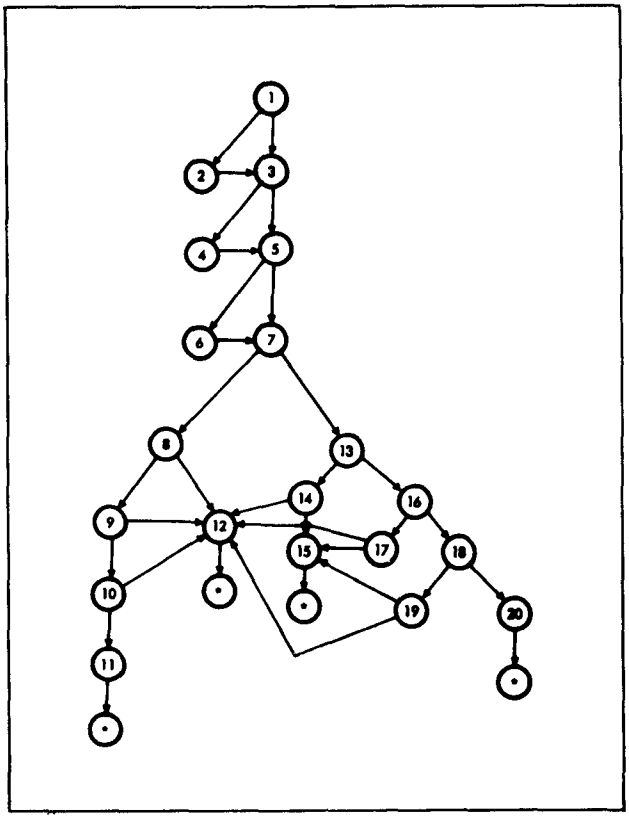

Figure 2. Network of Node-to-Node Branching Potential

Identification Number	Path Representation
P1	1-2-3-4-5-6-7-13-16-18-20
P2	1-2-3-5-7-13-14-12
P3	1-2-3-5-7-13-14-15
P4	1-3-4-5-7-13-16-17-12
P5	1-3-4-5-7-13-16-17-15
P6	1-3-5-6-7-13-16-18-19-12
P7	1-3-5-6-7-13-16-18-19-15
P8	1-3-5-7-8-9-10-11
P9	1-3-5-7-8-9-10-12
P10	1-3-5-7-8-9-12
P11	1-3-5-7-8-12

Figure 3. Logical Paths Through the Program

	Input Values		
Path ID	I	J	K
P1	1	1	1
P8	2	3	4
P3	2	2	3
P4	1	2	1
P6	2	1	1
P2	1	1	2
P5	2	3	2
P7	3	2	2
P9	1	3	2
P10	3	2	1
P11	1	2	3

Figure 4. Test Case Input Values for Path Operation

Subsets of Paths to Guarantee Usage of All Statements:

P1, P8, P3, P4, P6

Subset of Paths to Guarantee Usage of All Branches:

P1, P8, P3, P4, P6, P2, P5, P7, P9, P10, P11

Figure 5. Path Subsets for Program Testing Coverage

Now let's suppose that there is an actual, ultimate user of TTDP; that is, there exists a person somewhere who awaits completion of TTDP program development and testing so that the program can be put to work solving real triangle type determination problems. Let's further assume:

1) the user has been doing triangle type determination by hand for quite a while and, therefore, has a good idea of the general types of problems to which the program will be exposed during operational usage, and

2) the user has requested, paid for and used software in the past and, in view of some unpleasant experience, now seeks a convincing argument and some assurance that TTDP will not fail to meet the specified needs in the operational environment.

Structural element testing strategy. The dominant theme in Reference 5 focused on techniques for developing a set of test cases which served to accomplish a certain (albeit limited) objective, i.e., assurance that each and every structural element would be exercised at least one time during execution of the program with the complete set of test cases. Preparation of the test cases was influenced by only one factor which was in any way descriptive of expected operational usage of the program; i.e., it was assumed that the program would be presented with only positive integer values.

It is interesting, but perhaps a little disturbing, to consider what kinds of statements can be made about the reliability of TTDP upon successful execution of the test cases. One might be prompted, for example, to state: "The program has been tested with 5 (11) test cases as required to cause all statements (branches, paths) to be exercised at least once with selected inputs. The program operated successfully producing correct results for all 5 (11) test cases; thus, the best estimate of the probability of failure during operational usage is 0 and the probability of successful operation (reliability) is 1." This statement is, of course, not much better than the age-old: "It worked OK for me, so it ought to work OK for you."

It is possible to add some punch to the statement through systematic application of the techniques described in the preceding section. We assumed, above, that the user was intimately familiar with the problems which TTDP is supposed to handle. If we look for ways in which expected operational usage might be characterized, there are several obvious ones that serve our immediate purpose. Suppose, for instance, that the user reflected for a while on past experience with triangle type determination and was able (with some reasonable degree of confidence) to state:

> About half of the time, the three integers do not represent a triangle. About one-third of the time, the triangle is isosceles; one-tenth of the time, it's scalene. The rest of the time it turns out to be equilateral.

This statement, in effect, defines a four-way, functional partition of the input domain such that:

G_1 = set of all combinations of positive integer values which do not form a triangle

G_2 = set of all combinations which form an isosceles triangle

G_3 = set of all combinations which form a scalene triangle

G_4 = set of all combinations which form an equilateral triangle

Furthermore, if it is assumed that past experience is representative of expected future TTDP usage, then we may assign fractional values indicating the probability that a given combination of integer values will belong to one of the sets as follows:

$P(G_1) = .5$

$P(G_2) = .333$

$P(G_3) = .1$

$P(G_4) = 1 - [P(G_1) + P(G_2) + P(G_3)] = 1-.933 = .067$

The set of P values is one instance of the operational profile probability distribution treated at length in the earlier discussion.

It is a relatively simple matter now to examine the actual test case input values (Figure 4) and corresponding output results and classify each test according to the set, G_k, to which it belongs. Doing so for the five test cases (Figure 5) yields the following frequencies:

$$n_1 = 2,\ n_2 = 1,\ n_3 = 1,\ n_4 = 1$$

Applying the previously discussed χ^2 formulation, we obtain:

$$\chi^2 = \sum_{k=1}^{4} \frac{[n_k - nP(G_k)]^2}{nP(G_k)}$$

$$\chi^2 = \frac{[2-5(.5)]^2}{5(.5)} + \frac{[1-5(.333)]^2}{5(.333)} + \frac{[1-5(.1)]^2}{5(.1)} + \frac{[1-5(.067)]^2}{5(.067)}$$

$$\chi^2 = .1 + .265 + .5 + 1.32$$

$$\chi^2 = 2.185$$

where the χ^2 value is a simple measure of the degree to which the actual test cases are representative of the defined operational profile probability distribution.

Similarly, we may examine the set of 11 test cases which are necessary to exercise all branches and logical paths derivable from the program (Figures1 and 2) and listed in Figure 3. Review of test results with reference to Figures4 and 5 yields:

$$n_1 = 6,\ n_2 = 3,\ n_3 = 1,\ n_4 = 1$$

and

$$\chi^2 = \frac{[6-11(.5)]^2}{11(.5)} + \frac{[3-11(.333)]^2}{11(.333)} + \frac{[1-11(.1)]^2}{11(.1)} + \frac{[1-11(.067)]^2}{11(.067)}$$

$$\chi^2 = .045 + .120 + .009 + .094$$

$$\chi^2 = .268$$

Finally, a direct comparison of the χ^2 values (i.e., 2.185 versus .268) causes us to conclude that the set of 11 test cases is more representative of expected operational usage than is the set of five test cases. On the other hand, we tend to be less critical of the five cases if we consider another five-case set consisting of all equilateral triangles for which the resulting χ^2 value is approximately 69.5. Similarly, a set of 11 equilateral triangle test cases would yield a whopping big χ^2 of 153.2 making the measured value of .268 look very good. In fact, a little more arithmetic makes us feel even better when it's seen that the minimum χ^2 value achievable with 11 test cases is .179 [since it is impossible to obtain $n_k = 11 \cdot P(G_k)$ for all k].

If we now enter a table of χ^2 with three degrees of freedom, we find that the probability of obtaining a value of χ^2 as large as or larger than .352 is .95. Since the observed value of .268 falls close to the tabled value of .352, the probability of the obtained value is slightly greater than .95. Similarly, the probability of the observed value of 2.185 is slightly greater than .5. We are now ready to return to the previous statement about the reliability of TTDP and modify the last sentence as follows:

> "The program operated successfully producing correct results for all 11 test cases; thus, the assessed failure ratio is 0/11 and the assessed reliability is
>
> $$\hat{R}_2 = 1 - \frac{0}{11} = 1.$$

The best estimate of the probability of success during operational usage is derived from:

$$\hat{R}_1 = 1 - \sum_{k=1}^{4} \frac{f_k}{n_k} P(G_k)$$

where $P(G_k)$ is the probability assigned to k^{th} set of the four sets constituting the operational profile, n_k is the number of test cases belonging to the k^{th} set, and f_k is the number of test cases (from the k^{th} set) which failed. Thus, the predicted operational usage reliability, R_1, is easimated as:

$$\hat{R}_1 = 1 - \left[\frac{0}{6}(.5) + \frac{0}{3}(.333) + \frac{0}{1}(.1) + \frac{0}{1}(.067)\right] = 1$$

i.e., $\hat{R}_1 = 1$ with confidence slightly greater than .95. Similarly, from the set of five test cases, $\hat{R}_1 = 1$ with confidence slightly greater than .5."

It is worth a little more effort to see what happens if not all of the test cases were successful. In actual practice, it may well be required that all test cases produce satisfactory results. For purposes of discussion, however, let's assume that one of the 11 test cases (e.g., one of the isosceles sets of integer inputs) causes TTDP to fail and, for one reason or another, the problem has not been fixed. Obviously, the assessed failure ratio would then be 1/11 and the assessed reliability would be $\hat{R}_2 = 1 - 1/11 = .909$. The predicted operational usage reliability, however, is obtained by computing:

$$\hat{R}_1 = 1 - \left[\frac{0}{6}(.5) + \frac{1}{3}(.333) + \frac{0}{1}(.1) + \frac{0}{1}(.067)\right] = .889$$

Thus, $\hat{R}_1 = .889$ with confidence slightly greater than .95.

Notice, however, that if the single failure had come from the equilateral triangle test case, we would still obtain

$$\hat{R}_2 = 1 - \frac{1}{11} = .909,$$

but the predicted operational usage reliability would be higher, i.e.,

$$\hat{R}_1 = 1 - \frac{1}{1}(.067) = .933$$

with unchanged confidence slightly greater than .95 based on the measure of the representativeness of the accomplished testing.

The preceding discussion and derivation exemplifies one approach to improving upon an otherwise uncertain statement about the operational reliability of TTDP. Note that the choice of the particular approach and the definition of the functional partition of the input domain was prompted directly by the form of the user's statement about expected operational usage. The statement itself was based upon 1) a good deal of past experience with triangle type determination and, 2) the user's ability to characterize that experience in a quantified, meaningful way.

We may approach the problem in a slightly different manner by exploring at least one alternate form of the user's statement. For example, let's assume that the integer values represent rank scores of a student on a series of three separate examinations. To simplify our computations, let's also assume that the rank scores range from 1 to 4. Now if we define max as the maximum difference between two of the three scores, i.e.:

$$\max = \text{maximum}\left[|I-J|, |I-K|, |J-K|\right]$$

then the following operational usage partition of the input domain is possible:

S_1 = set of all combinations of rank scores for which max = 0

S_2 = set of all combinations of rank scores for which max = 1

S_3 = set of all combinations of rank scores for which max = 2

S_4 = set of all combinations of rank scores for which max = 3

Furthermore, if the user knows enough about the similarity of the three examinations and is aware of the tendency of rank scores to "bunch up" accordingly, it is entirely possible that the following operational profile probability distribution could be established.

$$P(S_1) = .3,\ P(S_2) = .4,\ P(S_3) = .2,\ P(S_4) = .1$$

As in the preceding derivation, an examination of the 11 test cases used to exercise all paths of TTDP yields the following frequencies:

$$n_1 = 1,\ n_2 = 6,\ n_3 = 4,\ n_4 = 0$$

We proceed as before and compute the χ^2 value as:

$$\chi^2 = \sum_{k=1}^{4} \frac{[n_k - n\,P(S_k)]^2}{nP(S_k)}$$

$$\chi^2 = \frac{[1-11(.3)]^2}{11(.3)} + \frac{[6-11(.4)]^2}{11(.4)} + \frac{[4-11(.2)]^2}{11(.2)} + \frac{[0-11(.1)]^2}{11(.1)}$$

$$\chi^2 = \frac{(2.3)^2}{3.3} + \frac{(1.6)^2}{4.4} + \frac{(1.8)^2}{2.2} + \frac{(1.1)^2}{1.1}$$

$$\chi^2 = 1.603 + .582 + 1.473 + 1.1$$

$$\chi^2 = 4.758$$

From the table of χ^2 values (again using 3 df), we find that the observed value of χ^2 falls close to the table value of 4.642. The probability of obtaining a value of χ^2 as large as or larger than 4.642 is .2, and we may modify the earlier proposed and once-modified statement about the reliability of TTDP accordingly. That is, in the event that all 11 test cases executed successfully, and substituting S for G,

we observe $\hat{R}_2 = 1$

and predict $\hat{R}_1 = 1$ with confidence slightly less than .2.

In the event that one of the test cases failed (e.g., one of the four belonging to S_3) and assuming that 0/0 = 0, we obtain:

$$\hat{R}_2 = 1 - \frac{1}{11} = .909 \text{ (same as before)}$$

but

$$\hat{R}_1 = 1 - \sum_{k=1}^{4} \frac{f_k}{n_k} P(S_k)$$

$$\hat{R}_1 = 1 - \frac{0}{1}(.3) + \frac{0}{6}(.4) + \frac{1}{4}(.2) + \frac{0}{0}(.1)$$

$$\hat{R}_1 = 1 - 0 + 0 + .05 + 0 = .95$$

Thus, the predicted operational usage reliability, $\hat{R}_1$, is estimated as .95 with confidence slightly less than .2.

Before leaving the example, it should be pointed out that there is a straightforward way to improve (i.e., decrease the χ^2 value and increase the confidence) upon the current situation through strategic addition of test cases to the original set of 11. Notice, for example, that the largest contribution to the above χ^2 value resulted from a noticeable difference between the observed frequency and expected frequency of test cases belonging to S_1. By simply adding one test case satisfying the property max = 0, we obtain:

$$\chi^2 = \frac{[2-12(.3)]^2}{12(.3)} + \frac{[6-12(.4)]^2}{12(.4)} + \frac{[4-12(.2)]^2}{12(.2)} + \frac{[0-12(.1)]^2}{12(.1)}$$

$$\chi^2 = .711 + .300 + 1.067 + 1.2$$

$\chi^2 = 3.278$ with corresponding confidence slightly greater than .3

Noticing that the largest contribution now comes from the fourth term we may add a 13th test case (satisfying max = 3) which yields:

$$\chi^2 = \frac{1}{13}(12.033 + 1.6 + 9.8 + .9) = 1.872$$

with corresponding confidence of approximately .6.

Clearly, the process can be continued as long as necessary, thus ensuring that the evolving set of tests becomes sufficiently representative of expected operational usage to permit prediction of operational reliability with specified (and perhaps required) confidence.

Conclusions and recommendations

We have provided both a detailed description and a demonstrated application of a methodology which relates software testing strategy to reliability assessment. Considerable attention was given to presenting the methodology in a way which would both encourage and facilitate its application. As indicated earlier, the representative testing and reliability measurement methodology is but one of the products of a substantial on-going research activity. The research to date has focused on the fundamental properties of software, of classical and innovative testing technology, and of the basic requirements for highly confident estimation of software reliability.

The methodology as described is not limited in application to small and trivial programs such as the triangle type determination example. It is not yet clear, however, just what kind of effort or how much of it is required for effective application to larger and more complex programs. There is, indeed, a great deal more to be learned, and continued research along the following lines is considered to be essential:

- Investigation of difficulties and development of techniques for deriving and expressing operational usage profiles; i.e., identification of appropriate partitions of a program's Input Data Space and development of corresponding probability distributions as a function of software performance requirements specifications and other available documentation, if any.

- Definition of the responsibilities of the typical participants in the software procurement process as necessary to ensure effective application of the methodology. For example, investigate candidate approaches to user specification of expected operational usage, and establishment of minimum acceptable levels of 1) testing representativeness, 2) assessed and predicted operational usage reliability, and 3) confidence in the reliability estimate.

Determination of potential cost, quality and schedule impact of operational usage testing with respect to current methods of retesting following software modification; e.g., investigate the application of the chi-square test for representativeness as an indication of retesting sufficiency and identify possible trade offs between reduction in operational usage due to selected retesting of modified software elements.

Implicit in these statements of required research are many as yet unanswered questions. In some respects this paper has only scratched the surface of the software reliability problem. It is hoped, however, that sufficiently detailed and careful treatment has been given to the representative testing and reliability measurement methodology to stimulate the reader's interest and to focus research on one or more of these unresolved issues.

References

1. Nelson, E.C., "A Statistical Basis for Software Reliability Assessment," TRW Software Series SS-73-03, March 1973.
2. Brown, J.R. and Buchanan, H.N., "The Quantitative Measurement of Software Safety and Reliability," TRW Report SDP 1776, 24 August 1973.
3. Krause, K.W., Smith, R.W. and Goodwin, M.A., "Optimal Software Test Planning Through Automated Network Analysis," Proceedings IEEE Symposium on Computer Software Reliability, 1973, pp. 18-22.
4. Lipow, M., "Application of Algebraic Methods to Computer Program Analysis," TRW Software Series SS-73-10, May 1973.
5. Brown, J.R., "Practical Applications of Automated Software Tools," WESCON 1972, Session 21 and TRW Software Series SS-72-05, September 1972.
6. Gruenberger, F., "Program Testing: The Historical Perspective," Program Test Methods, Ed., W.C. Hetzel, Prentice-Hall, 1973, pp. 11-14.