The Cost of Data Flow Testing: An Empirical Study

ELAINE J. WEYUKER

Abstract—A family of test data adequacy criteria employing data flow information has been previously proposed, and theoretical complexity analysis performed. This paper describes an empirical study to determine the actual cost of using these criteria. This should help establish the practical usefulness of these criteria in testing software, and serve as a basis for predicting the amount of testing needed for a given program.

Index Terms—Data flow testing, software testing, test data adequacy.

I. INTRODUCTION

FOR the last several years, we have been developing, refining, and experimenting with a theory of test data selection and evaluation based on the use of data flow information in the subject program [13], [14], [17], [6], [3], [4], [5], [12]. Although these strategies were originally intended to be applied at the module level during unit testing, preliminary work has been done to extend these techniques for use in integration testing and regression testing [12]. Theories with a similar underlying philosophy were developed by Herman [7], Ntafos [10], and Laski and Korel [9], and comprehensive comparisons of these criteria were presented by Clarke et al. in [1] and by Ntafos in [11].

Our data flow testing criteria require the selection of test data that test certain paths from a point in a program where a variable is defined, to points at which that variable definition is subsequently used. By varying the required combinations of definitions and uses, a family of test data selection and adequacy criteria was defined.

Every variable occurrence in the program is assigned a category. Besides distinguishing between definitions (variable occurrences at which a variable is given a new value, as in an input statement or the left-hand side of an assignment statement) and uses (variable occurrences at which a variable is not given a new value), we distinguish between two distinct types of uses: predicate uses (p-uses) and computation uses (c-uses). P-uses are uses in the predicate portion of a decision statement such as while . . . do, if . . . then . . . else, or repeat . . . until statements. C-uses are uses which are not p-uses, including v variable occurrences in the right-hand side of an assignment statement, or an output statement.

The criteria require that test data be included which cause the traversal of subpaths from a variable definition to either some or all of the p-uses, c-uses, or a combination of p-uses and c-uses of that variable. For example, the all-definitions criteria requires sufficient test data such that every definition is used at least once. Put another way, this criterion requires that test data be included which cause the traversal of at least one subpath from each variable definition to some p-use or some c-use of that variable definition. The all-c-uses criteria requires that test data be included which cause the traversal of at least one path from each variable definition to every c-use of that variable definition. The all-p-uses criteria requires that test data be included which cause the traversal of at least one path from each variable definition to every p-use and every c-use of that variable definition. The all-du-paths criteria requires that test data be included which cause the traversal of every simple subpath from each variable definition to every p-use and every c-use of that variable definition. The relationships between these criteria were explored in [13] and [14]. Precise definitions of the entire family of criteria are included in [14] for a very simple language, and in [5] for Pascal.

After originating the theory, and proving that, at least from a theoretical point of view, the criteria have desirable characteristics, we considered (theoretical) complexity issues. We demonstrated in [17] that, in the worst case, all but one of the criteria require at most a number of test cases which is quadratic in the number of decision statements of the module being tested. We further demonstrated that these upper bounds were "tight" by exhibiting programs which actually require the specified worst case number of test cases. But examination of these "worst case programs" indicated that they are not "natural" programs and we hypothesized that "real" programs would require far fewer test cases to satisfy the data flow criteria.

Encouraged by the positive theoretical results and complexity bounds, we applied our theory to test many programs in order to get a feel for its usefulness in detecting errors and evaluating the comprehensiveness of proposed test sets. Since this was done using hand simulation, most programs in these informal experiments were of necessity small. This experience was also promising, and we therefore built a tool, ASSET, to automate the application of these techniques.

0098-5589/90/0200-0121$01.00 © 1990 IEEE
The initial version of ASSET [6], [3] accepted programs written in the very limited subset of Pascal of the original theory [13], [14]. We have now extended the theory to full Pascal [5], and currently ASSET accepts programs written in (almost) full Pascal. The theory has also been extended recently to be used for regression testing [12], and work is currently being done to provide a dataflow regression testing facility in ASSET.

Having implemented ASSET, and extended the theory of dataflow testing to include Pascal, rather than the original "toy" language, we have now gained a great deal of experience from running ASSET on many sample programs. This experience strengthened our conviction that using dataflow information as the basis of a test data adequacy criterion is useful and sound. At this point we felt that systematic empirical studies to evaluate the dataflow testing criteria were essential.

In this paper we present the results of an empirical study which was designed to investigate the costs of using various dataflow testing techniques in practice. There are two main goals of this study. The first is to confirm our intuition and informal experience that our family of software testing criteria based on the program's dataflow characteristics is practical to use. That is, we gathered evidence that even as the program size increases, the amount of testing, expressed in terms of the number of test cases sufficient to satisfy criterion, remains modest. We explore, in this study, several ways of evaluating this hypothesis.

The second goal is closely related to the first: to provide the prospective user of these criteria with a way of predicting the number of test cases that will be needed to satisfy a given criterion for a given program. This provides testers with a basis for selecting the most comprehensive criterion that they can expect to satisfy. Again we consider several plausible bases for such a prediction.

II. THE DESIGN OF THE STUDY

One difficulty in designing software engineering experiments in general, and testing experiments in particular, is choosing appropriate software to be studied. As an academic, it is tempting to use the available captive audience, namely students, to provide the software. However, we believe strongly that this is inappropriate for several reasons.

First, the programs should be "real." That is, they should not have been written solely for the experiment. One cannot expect the same care to be taken for a "throwaway" program as for a production program. A second weakness of student-written programs is that students are generally apprentices. Determining the usefulness of a testing theory for such programmer "trainees" does not necessarily give us much useful information about its utility for professional programmers. A third problem involves scale. Student programs are generally small and self-contained. Do the techniques scale up?

For these, and related reasons, we decided that our empirical experiments would not be done using student-written software. Therefore, the next question to decide was: where will the software come from? Since we wanted "real" software, that was written by professional programmers, and of a substantial size, we selected the suite of programs, Software Tools in Pascal [8] by Kernighan and Plauger, which fit all of these characteristics and is readily available in machine readable form. In particular, the suite consists of over 100 routines. Since we wanted to establish "how expensive" these dataflow criteria are to use, many modules had to be tested to determine how many test cases were needed to satisfy each of the criteria in practice. This suite of programs was therefore the basis of our experiments. We tested most of the modules in the suite, using most of the criteria of our family, in order to establish empirical complexity estimates.

Testers were instructed to select test cases using the selection strategy of their choice, other than a dataflow criterion. Each tester's goal was to do a "good job" of testing and to apply the dataflow criteria to evaluate the adequacy of the test set once they believed they were done testing.

Four people worked as testers. Three of the testers had had industrial experience; the fourth was a graduate student with little experience other than undergraduate programming exercises. Although he was bright and eager, this lack of experience was an obvious liability and only one module tested by this person is included in the study.

We also decided in this study that no attempt would be made to minimize the number of test cases used in each test set. Instead, "atomic" test cases were selected. That is, rather than attempting to cleverly select a test case which might fulfill several characteristics simultaneously, testers were instructed to select "natural" test cases, each with a single purpose. Thus, for example, rather than selecting a test case which begins with at least two blank characters, is at least ten characters long, containing at least one repeated nonblank character, given that these three characteristics are determined to be significant, we would instead select three distinct test cases, each displaying only one of the characteristics. This was done because testing practitioners frequently do not attempt to minimize test sets. In fact if regression testing is to be performed, it can be argued that having each test case have a single, clearly identifiable role is desirable. Also, the use of "atomic" test cases yields pessimistic results. We felt this was important since if nonminimal test sets which satisfy the dataflow criteria were of manageable size, then the criteria are certainly practical to use.

Although ASSET was applied to almost all of the subprograms in the Kernighan and Plauger suite, we considered in this study only those subprograms with five or more decision statements. There were 29 such programs used in this study which are listed in the Appendix. For programs with fewer decision statements, the upper bounds on the number of test cases needed to satisfy the criteria are so small that there is no difficulty running that number of test cases. Only as the number of decision statements grows does it become important to determine
whether substantially fewer test cases suffice in practice to satisfy the criteria.

If the selected criterion requires the traversal of unexecutable subpaths, then no test set, regardless of how extensive, will satisfy the criterion and thereby adequately test the program. In such a case, we shall say that a test set 'almost satisfies' a data flow criterion provided it causes all the executable subpaths required by the given criterion to be traversed. In [4], [5], we introduced a new family of accuracy criteria, known as the feasible data flow criteria, which are based on the data flow criteria but require only that executable definition/use associations be exercised. Since it is clearly impossible to cause unexecutable paths to be executed, we consider a criterion satisfied when a test set almost satisfies the criterion. In the remainder of this paper, we shall say that a criterion has been satisfied provided that it has been almost satisfied.

III. EXPERIMENTAL RESULTS

The primary questions addressed in this study involve the 'real' complexity of the data flow testing criteria, as assessed in a variety of ways. In [17] we demonstrated the following theoretical upper bounds on the number of test cases required to satisfy each of the data flow criteria. If \( d \) is the number of (two-way) decision statements in the program, then all-c-uses, all-p-uses, and all-uses require at most \( d^2 + 4d + 3 \) test cases, and all-du-paths requires at most \( 2d^2 \) test cases. Furthermore, for each of these bounds other than all-c-uses, we constructed a program for which the only test sets which can satisfy the selected criterion are at least the size of the stated upper bound. Of course, test sets may exceed the size of the upper bound and still not satisfy a given criterion if the test cases have not been judiciously selected or if there are unexecutable definition/use associations.

In this study we consider the 'empirical complexity' for the criteria all-c-uses, all-p-uses, all-uses, and all-du-paths. We did not consider the requirements for the all-definitions criterion, except in comparison to the more expensive criteria, since for this criterion the upper bound on the number of test cases is quite moderate. Since the theoretical upper bounds were expressed in terms of the number of decision statements in the program, we have used that as our basis for the empirical study as well.

Given the previously stated goals of this study, for each of the 29 programs in the Kernighan and Plauger suite and each of the considered data flow criteria, we collected data and computed:

1) The least squares line: \( t = \alpha d + \beta \), where \( t \) is the number of test cases sufficient to satisfy the given criterion for the subject program, and \( d \) is the number of decision statements in the subject program.

2) The weighted average of the ratios of the number of decision statements in a subject program to the number of test cases sufficient to satisfy the selected criterion.

3) The maximum value of the ratio of the number of test cases sufficient to satisfy the selected criterion for a given subject program to the number of decision statements in that program.

4) The weighted average of the ratios of the theoretical upper bound on the number of test cases needed to satisfy the selected criterion for a given subject program to the number of test cases sufficient to satisfy the selected criterion.

5) The weighted average of the ratios of the number of test cases sufficient to satisfy the all-definitions criterion to the number of test cases sufficient to satisfy the all-uses criterion.

6) The weighted average of the ratios of the number of test cases sufficient to satisfy the all-definitions criterion to the number of test cases sufficient to satisfy the all-du-paths criterion.

7) The weighted average of the ratios of the number of test cases sufficient to satisfy the all-uses criterion to the number of test cases sufficient to satisfy the all-du-paths criterion.

One way to assess the 'empirical complexity' of the various data flow criteria is to use the method of least squares. We consider points \( (d_i, t_i) \) where \( d_i \) denotes the number of decision statements in program \( i \), and \( t_i \) denotes the number of test cases used to satisfy the given criterion for program \( i \), and computed the least squares line for each criterion. These results are shown in row 1 of Table I.

Notice that in each case the coefficient of \( d \) is less than 1, reinforcing our intuition that the relationship between the number of test cases and the program size is truly a linear one.

In addition to least squares analysis, we also directly compared the number of test cases sufficient to satisfy a given criterion to the number of decision statements in the program. This comparison provided an 'empirical average case' complexity for each criterion.

We computed, for each program, the ratio of the number of decision statements in the program to the number of test cases sufficient to satisfy the given criterion, and then found the average of these ratios. That is, if \( t \) denotes the number of test cases which satisfy the selected criterion for program \( i \), and \( d_i \) denotes the number of (two-way) decision statements in program \( i \), we first computed \( s_i = d_i / t_i \) for each \( i \). We then computed the average value:

\[
E = \frac{1}{n} \sum_{i=1}^{n} s_i,
\]

where \( n \) is the number of programs. We then expressed this weighted average as \( t = (1/E)d \). The results for the four criteria studied are shown in row 2 of Table I.

As was the case with the least squares analysis, this empirical average case analysis indicates that in a practical sense, each of these criteria can be viewed as being linear with a small coefficient, rather than being quadratic or exponential as the theoretical upper bounds state.

It is also interesting that the coefficient of \( d \), using the least squares analysis and empirical average case analysis are quite similar for each of the criteria. This implies that the two analysis will give the tester similar predictions for
the number of test cases needed to test a given size program.

Both the least squares line and the computation of the “empirical average value” of \(d/E\) for a given criterion provide a rough guide for predicting the amount of testing that will be sufficient for a given program with \(d\) decision statements. This permits testers to select the most comprehensive criterion their budget permits. For example, the least square line predicts that a test set of roughly half the number of decision statements in the program will suffice to satisfy the all-c-uses criterion, while the computation of \(E\) predicts that somewhat fewer test cases will suffice to satisfy the same criterion. Thus a person testing a subprogram with 15 decision statements might expect to need roughly 6 to 10 test cases to satisfy the all-c-uses criterion, based on our empirical results. For the all-p-uses criterion, our results indicate that the tester will need roughly 11 or 12 test cases for a subprogram with 15 decision statements. For the all-uses criterion, the tester would expect that a test set of size 11 to 13 would suffice to test the 15 decision statement subprogram, and similarly, we would expect that a test set of size 12 to 15 would suffice to test the subprogram using the all-du-paths criterion. The fact that the tester can expect to need a very modest number of test cases in order to do quite a comprehensive job of testing is especially surprising and encouraging for the all-du-paths criterion which requires in the worst case a number of test cases which is exponential in the number of decision statements in the program.

It is also important to consider the minimum value of \(d_i/d\) (or equivalently the maximum value of \(t_i/d\)) encountered for each criterion in the study. Even though the average case value is moderate, it is important to determine whether there were any cases encountered which required exceptionally large numbers of test cases. Practically, this ratio represents an empirical worst case for each of the data flow criteria considered.

For the 29 programs of the Kernighan and Plauger suite, the maximum value of \(t_i/d\) for the criterion all-c-uses, was 3.5. For the all-p-uses criterion, the maximum value was 2.33, for the all-uses criterion the maximum value was 3.67, and for the all-du-paths criterion the maximum value was 3.67. Note that in each case, this ratio was small, giving us further evidence that the data flow criteria are usable in a practical setting. It indicates that in addition to being linear in the number of decision statements in the subprogram in the “empirical average case,” each of these criteria can be viewed as being linear in the number of decision statements in the “empirical worst case” as well.

An interesting question is whether we can characterize, at least informally, those programs which lead to these empirical worst cases. We shall discuss this question in a later section. It is worth noting here, however, that in this study, the same program was responsible for the empirical worst cases for each criterion. That is, for each of the ways of assessing adequacy which we considered, the same program required the greatest amount of testing relative to its size.

Another way to consider the data is to compare the actual number of test cases sufficient to satisfy criteria to the theoretical upper bounds. For each program, we computed the ratio of the upper bound for that program to the number of test cases which satisfy the given criterion, and found the average of these ratios. That is, letting \(r_i\) denote the number of test cases which satisfied the selected criterion for program \(i\), and \(b_i\) denote the upper bound for program \(i\), we first computed \(r_i = b_i/t_i\) for each \(i\). Taking a weighted average of these values, we computed:

\[
A = \frac{1}{n} \sum_{i=1}^{n} r_i
\]

where \(n\) is the number of programs. Row 4 of Table I shows the value of \(1/A\) for each of the four criteria considered.

Notice that even for programs of moderately large size (as assessed by the number of decision statements in the program), we found in practice that only a moderate number of test cases were needed for each of the criteria.

The final question we considered was: “How much more expensive is it, in practice, to use the more demanding criteria, rather than the weaker criteria?” The answer, surprisingly, was that, on the average, it was not much more difficult to satisfy the most demanding criteria as compared to the least demanding criterion. To determine this, we considered, for each subprogram of the study, the ratio of the number of test cases sufficient to satisfy the all-definitions criterion to the number of test cases sufficient to satisfy all-uses, as well as the ratio of the number of test cases sufficient to satisfy the all-definitions criterion to the number of test cases sufficient to satisfy the all-du-paths criterion. Finally, we considered the ratio of the number of test cases sufficient to satisfy the all-uses criterion to the number sufficient to satisfy the all-du-paths criterion. Recall that the all-uses criterion has a theoretical upper bound which is quadratic in the number of decision statements in the program, whereas the all-du-paths criterion has a theoretical upper bound which is exponential in the number of decision statements. We let \(f_i\) denote the number of test cases sufficient to satisfy the all-definitions criterion for program \(i\), \(u_i\) denote the number of test cases sufficient to satisfy the all-uses criterion for program \(i\), and \(p_i\) denote the number of test cases sufficient to satisfy the all-du-paths criterion for program \(i\). Then we computed:

\[
W_i = \frac{1}{n} \sum_{i=1}^{n} \frac{f_i}{u_i}
\]
\[ W_j = \frac{1}{n} \sum_{i=1}^{n} f_i \]
\[ W_j = \frac{1}{n} \sum_{i=1}^{n} p_i \]

These results are presented in Table II. In each case we see that if the cost of testing is assessed in terms of the number of test cases sufficient to satisfy the criterion, it is hardly more expensive to satisfy a very demanding and comprehensive criterion than a much weaker one. This was quite a surprising and encouraging result.

### IV. Additional Data

In this section we present some additional data and discuss the implications. Shimell and Leveson [16] and Shimell [15] describe experiments to compare the effectiveness of various validation techniques. He applied the all-p-uses criterion to a suite of eight numerical programs as part of a study to compare the effectiveness of various testing techniques. The specification for these programs was derived from a specification provided by TRW. AST-SET was used to assess the coverage attained by proposed test sets. The programs were numerical in nature, and ranged in size from 1186 to 2489 lines of Pascal code.

Again, no attempt was made to minimize the number of test cases used. Table III shows the data for each of the eight programs as well as the number of required test cases predicted by the least squares line and the weighted average, as determined by the Kernighan and Plauger bound. All data is in the theoretical upper bound, and the ratios of the size of the test set sufficient to satisfy the all-p-uses criterion to the various predicted test set sizes.

Notice that in every case, the number of test cases sufficient to satisfy the all-p-uses criterion for these programs was substantially smaller than the number predicted by our empirical study using the Kernighan and Plauger suite, as assessed by either the least squares or the weighted average measure. In addition, in each case the test sets were an extremely small fraction of the theoretical upper bound, none exceeding 1 percent of this bound.

We computed the least squares line using the Shimell data and found it to be:

\[ t = 0.02d + 104.28 \]

Since each of the eight programs in this set could be tested using the all-p-uses criterion with test sets of almost equal size (103 to 117 test cases) even though the number of decision statements in the programs varied substantially (196 to 434), we believe that the least squares line determined by this data would not be applicable to substantially different sized programs.

The empirical average case for these programs using the all-p-uses criterion was found to be:

\[ t = 0.39d. \]

| \# revisions | \# test cases | predicted least squares | \# test cases | \# test cases | predicted | weighted | \|/\| | upper bound | \|/\||
|-------------|---------------|-------------------------|--------------|--------------|-----------|----------|------|
| 411         | 106           | 104                     | 78           | 72           | 108       | 107      | 0.02 |
| 209         | 100           | 100                     | 105          | 105          | 108       | 110      | 0.08 |
| 196         | 125           | 125                     | 127          | 127          | 127       | 127      | 0.12 |
| 225         | 110           | 110                     | 111          | 111          | 111       | 111      | 0.18 |
| 238         | 93            | 93                      | 94           | 94           | 94        | 94       | 0.22 |
| 183         | 100           | 100                     | 100          | 100          | 100       | 100      | 0.26 |
| 201         | 167           | 167                     | 168          | 168          | 168       | 168      | 0.24 |
| 123         | 98            | 98                      | 99           | 99           | 99        | 99       | 0.42 |

This, too, was substantially less than predicted by the Kernighan and Plauger suite data:

\[ t = 0.70d. \]

The empirical worst case encountered for the all-p-uses criterion for these programs was 104/173 = 0.60. This is in contrast to 2.33 encountered for the programs of the Kernighan and Plauger suite.

It is interesting to consider possible reasons why such small test sets sufficed for this set of programs. The programs were, on average, substantially larger than the programs in the Kernighan and Plauger suite. Although we expected that as the program size increased, the amount of test data needed to satisfy a given criterion would increase at a faster rate, this did not turn out to be the case. In fact, just the opposite appears to be the case—it was relatively easier to satisfy the all-p-uses criterion for the larger routines than for the smaller ones. Also, since the eight programs considered by Shimell and Shimell and Leveson were numerical and defined by a far more sophisticated specification than those in the Kernighan and Plauger suite, we would have expected the Shimell programs to be harder to test, as measured by the relative amount of test data needed to satisfy a given criterion. Again, this was not the case. At this time we cannot hypothesize why this is the case.

### V. Observations, Conclusions, and Future Work

#### A. Relations Among Criteria

In addition to the quantitative empirical results described above, a number of qualitative relationships also appear to hold. We noticed that the all-p-uses criterion was generally somewhat harder to satisfy than the all-c-uses criterion, and that a test set which satisfied all-p-uses usually satisfied all-c-uses too. This is in spite of the fact that all-p-uses and all-c-uses were shown to be independent criteria [13], [14]. This observation is supported by the values of \( E \) and the least squares lines for these two criteria.
We also observed that although all-us"es is a more demanding criterion than all-p"uses in the sense that every test set that satisfies all-us"es for a given program automatically satisfies all-p"uses for that program, a test set which was adequate when assessed by the all-p"uses criterion was generally also adequate using the all-us"es criterion. This observation was also supported by the values for \( E \) for the appropriate criteria.

We were surprised to observe that even though the all-du-paths criterion has an exponential upper bound whereas the all-us"es criterion has a quadratic upper bound, in practice test sets sufficient to almost satisfy all-us"es were frequently also sufficient to almost satisfy all-du-paths. From a practitioner's point of view, this means that using the all-du-paths criterion to assess adequacy may well be reasonable even when the program being tested is quite large. This observation was borne out by the value of \( E \) for all-du-paths and \( W \). Note that it was shown in [4], [5] that it is theoretically possible for a test set to almost satisfy the all-du-paths criterion without almost satisfying the all-us"es criterion, all-p"uses criterion, all-c"uses criterion, or all-definitions criterion. This is in contrast to the theorem proved in [14] that any test set which satisfies the all-du-paths criterion, satisfies the all-us"es criterion, and hence all of the other data flow criteria. The intuitive reason for this substantial difference between the data flow criteria and the feasible data flow criteria is that is is possible that there are definition/use associations which can only be executed by a path which traverses a loop several times. The technical details of this result are given in [5].

In practice, we rarely encountered this situation. That is, if a test set almost satisfied the all-du-paths criterion, it generally almost satisfied the weaker criterion as well.

B. Unexecutable Paths

Even though the size of the required test sets were not nearly as large as predicted by the theoretical upper bounds, we did encounter one practical difficulty when using the data flow criteria which has negative implications for the use of these criteria for large programs. The problem was determining which of the definition/use associations or du-paths were executable. This is a problem which is encountered when using many program-based criteria, including statement and branch coverage. It is particularly acute for the all-du-paths criterion since there are frequently a large number of unexecutable du-paths. In fact, we found that the unexecutable path problem, not the large number of required test cases, was the primary practical difficulty in using the all-du-paths criterion. Frequently, a test set which almost satisfies the all-us"es criterion for a given program also almost satisfies the all-du-paths criterion for that program, but there are many more unexecutable paths to contend with for the latter criterion.

Whenever it is determined that an adequacy criterion has not yet been satisfied, the tester must determine whether this is because the program has not yet been adequately tested or whether the criterion is not satisfiable. For any of the path selection testing criteria (including statement coverage, branch coverage, and the data flow criteria) there can be no algorithm to decide, in general, whether or not such a path segment is executable. Experience with ASSET has convinced us, however, that human beings are frequently very good at determining unexecutability. For example, it may be that in order for a given definition/use association to be executed, some switch must be set to "on" and "off" simultaneously, or some variable must have a value greater than 100 when it has just been initialized to 1. The intelligent tester recognizes that these conditions are impossible and correctly concludes that if these are the only associations that ASSET reports are still unexecuted, then the test set has effectively satisfied the selected criterion. Work has recently been done [2] to develop heuristics which can be used to help the tester determine whether selected definition/use associations or du-paths are executable. The manual application of these heuristics to several of the programs in the Kernighan and Plauger suite indicates that they are capable of correctly recognizing many of the unexecutable definition/use associations and du-paths. We therefore hope to incorporate these or similar techniques to automate the determination of unexecutable path segments and thereby further establish the usefulness of the data flow testing criteria and ASSET.

One helpful simple facility to assist the tester in dealing with the unexecutable path problem has recently been implemented. If a path segment or definition/use pair has been determined to be unexecutable, this program checks other unexercised paths to see if they contain any of the known unexecutable subpaths. Any path containing an unexecutable subpath is clearly unexecutable, and is automatically removed from consideration. This simple routine has made a significant difference in allowing the tester to handle this problem.

In at least one case, however, the tester was unable to make a definite determination of unexecutability. Although he was not able to devise a test case which caused the appropriate definition/use association to be exercised, he was also unable to show conclusively that the association was unexecutable. The tester was convinced, after substantial effort, however, that the association was unexecutable in the context of the calling program. That is, by analyzing the program, the tester believed that whenever this subprogram was called in the Kernighan and Plauger suite, the variables would be initialized in such a way that it was impossible to traverse the selected association. Nonetheless, the tester was unable to conclusively prove that this was the case. In a number of other cases, the tester was able to show conclusively that a path was unexecutable in context even though the path would be executable in other contexts.

C. Characterizing Programs by Testing Requirements and Type

We were interested in trying to characterize, at least informally, the types of programs which require the most
testing relative to their size (number of decision statements). As we saw, none of the subprograms studied really required a great deal of testing using any of the data flow criteria, and for a subprogram with $d$ decision statements, a maximum of $3.67d$ test cases were required, even for the most demanding criterion, all-du-paths. It is, of course, difficult to put programs into classes since relatively few programs were studied.

As mentioned above, we found that the same program accounted for the empirical worst case values for each of the data flow criteria. In addition, if that program were removed from consideration, and the remaining 28 programs examined to establish new empirical worst case bounds for the data flow criteria, we find again that a single program yielded the worst case values for all of the criteria. For this second program, the maximum ratios of test set size to number of decision statements was 2.4 for the all-c-uses criterion, 1.5 for the all-p-uses criterion, 2.4 for the all-uses criterion, and 2.5 for the all-du-paths criterion.

Interestingly, these two programs were the only two encountered in the Kernighan and Plauger suite which required substantially larger test sets than predicted by $E$ or the least squares analysis. Furthermore, they share another interesting characteristic. For each of these programs, it was substantially more difficult to satisfy the all-c-uses criterion than it was to satisfy the all-p-uses criterion, and these were the only two programs for which that was true. Certainly, this is a very small sample to serve as the basis for broad generalizations. Therefore, we plan to study more programs to see if we can explain this observed phenomenon. We also plan to determine whether this is simply a coincidence or whether programs which require substantially larger test sets to satisfy the all-c-uses criterion than the all-p-uses criterion, frequently have unusually high testing requirements for all the data flow criteria.

One limitation of using the Kernighan and Plauger suite as the basis for this empirical study is that a large percentage of the programs in the suite are similar in type: namely string processing programs. For this reason, we were very interested in the results for the all-p-uses criterion for the Shimeall programs since they were relatively large numerical programs. We would like to extend these experiments to include many more programs, with a greater variety of types of software. It would be interesting to see if there are substantially different results for other types of programs. It would be interesting to study the data flow criteria relative to database programs and determine whether there are substantially different results than obtained in this study and also to consider additional numerical programs and all of the data flow criteria for such programs.

A limitation of the Kernighan and Plauger suite of different nature is that they are well designed and highly modularized. As a result, almost all of the subprograms are relatively small in size. These are not, however, toy programs. We do not know, however, if substantially larger, poorly designed programs would generally require larger amounts of test data relative to their size than the Kernighan and Plauger programs required. Based on the preliminary results using the Shimeall programs, however, it does not appear that this is the case.

VI. Summary

We have found the results of this empirical study to be very encouraging and confirming of our intuition. In particular, we found that although the theoretical upper bounds on the number of test cases needed to satisfy most of the data flow criteria are quadratic or exponential, in practice only small numbers of test cases (as compared to the program’s size) were needed to satisfy the criteria. Additionally, our data provides the tester with a way of predicting how many test cases should be needed to satisfy a given criterion for a given program. It was also encouraging to see that using the most demanding criterion, all-du-paths, required only, on average, a factor of less than two times the number of test cases as the very undemanding criterion, all-definitions. All of these results provide evidence that the data flow criteria are usable in practice.

Of course, we have based our study on a set of programs which were written with particular emphasis on good programming style. We do not necessarily expect that our results will be applicable to all types, styles, and sizes of programs. For this reason, we were very interested in the limited results provided by Shimeall, and we are currently experimenting with a large suite of published numerical programs to see if the results for these programs are similar to those presented here. We are also using that suite of programs to determine the fault detection capabilities of the various data flow criteria. We expect to be able to classify faults and determine which data flow criteria are particularly well suited for detecting particular types of faults. In addition, we intend to compare the expected cost of using a given data flow criterion to its fault detection ability. Ultimately, we intend to compare the data flow testing criteria to other testing strategies, both from the perspective of cost of satisfying the criteria for a given program, and fault detecting ability.

Appendix

Programs Considered in the Study

AMATCH
APPEND
ARCHIVE
CHANGE
CKGLOB
CMP
COMMAND
COMPARE
COMPRESS
DODASH
EDIT
ENTAB
EXPAND
ACKNOWLEDGMENT

This study would not have been possible without the members of the Software Testing Group at Courant and the support of the National Science Foundation and the Office of Naval Research. In particular, ASSET was implemented largely by P. Frankl, S. Weiss, and E. Campbell. The test cases were designed and run by V. Barr, E. Campbell, B. Jeng, and P. Fryscak. T. Ostrand made many helpful suggestions both about the analysis of the data and the presentation of the results. T. Shimeall kindly provided the all-p-uses data for his program suite. I am indebted to all of these people.

REFERENCES


Elaine J. Weyuker received the M.S.E. degree from the Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, and the Ph.D. degree in computer science from Rutgers University, New Brunswick, NJ.

She is currently an Associate Professor of Computer Science at the Courant Institute of Mathematical Sciences, New York University, New York, NY. Before coming to NYU, she was a systems 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 a book (with Martin Davis), Computability, Complexity, and Languages, published by Academic Press.

Prof. Weyuker is currently the Secretary/Treasurer of ACM SIGSOFT and is on the editorial board of ACM Transactions on Software Engineering and Methodology. 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.