More Experience with Data Flow Testing

Elaine J. Weyuker

Abstract—Experience is provided about the cost and effectiveness of the Rapps–Weyuker data flow testing criteria. This experience is based on studies using a suite of well-known numerical programs, and supplements an earlier study using different types of programs. The conclusions drawn in the earlier study involving cost are confirmed in this study. New observations about tester variability and cost assessment, as well as fault detection, are also provided.

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

I. INTRODUCTION

In [18], we described a study involving the data flow testing criteria proposed in [16]. The goals of that study were to gain experience by assessing the cost, on average, of satisfying the data flow criteria and to provide the tester with a basis for predicting the size of a test set sufficient to satisfy a given criterion. Such results would give testers guidelines for estimating the most demanding testing criterion they could expect to be able to satisfy for a given amount of resources. For that study, we used programs from Kernighan and Plauger’s Software Tools in Pascal [10]. The results of the study were very encouraging, and are summarized in Section IV. Since most of the programs in [10] are well-structured, modularized, string processing programs, we wondered whether the results would be substantially different for other types of programs. One goal of the study described in the current paper, therefore, was to repeat the earlier study with another set of programs having substantially different characteristics.

Each study is relatively small and informal, and we do not argue the statistical significance of the results. They provide, however, indications of what a tester using these criteria might expect in terms of test set size. We also discuss in this paper, two additional studies. The first investigated the fault detection ability of these criteria and the second looked at variation in test set size due to differences between individual testers.

Fifteen programs from a suite of numerical programs were used for the study we now describe. These programs originally appeared in issues of the Communications of the ACM, and are available as a separate entity known as the Collected Algorithms from ACM [1]. In that form, each program is published with a brief specification and a description of all known bugs and proposed fixes. The collection contains numerical routines, written in ALGOL or FORTRAN. Since our data flow testing tool, ASSET, accepts only Pascal programs, each program was hand translated into Pascal on a statement by statement basis. Generally this involved only straightforward changes to account for minor syntactic differences between the languages.

In the next section we present informal definitions of the data flow testing criteria used. For a more formal introduction to the criteria, the original motivation for defining them, and relationships among them, the reader is referred to [16]. A preliminary version of ASSET is described in [5], and the earlier empirical study appears in [18].

Section III describes the design of the study. Section IV presents the results of our earlier study and the cost portion of the current study. Section V describes some additional cost data, and discusses some implications of these results. Section VI presents the fault detection portion of the study, and presents some observations about the meaningfulness of this data. Section VII discusses the effects of using different testers. Section VIII presents conclusions.

II. DATA FLOW TESTING

The Rapps–Weyuker [16] data flow testing criteria require the selection of test data that exercise 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. Other data flow criteria have been proposed in [9], [11], and [14]. Although each of these criteria has a similar underlying philosophy, each differs in the actual choice of definition and use combinations required. [2] and [15] investigated the relationships between the various data flow criteria. In this paper we restrict attention to evaluation of the Rapps–Weyuker data flow criteria, and refer to them simply as data flow testing.

We begin by assigning a category to every variable occurrence in the program. Besides distinguishing between definitions (variable occurrences at which a variable is given a new value) and uses (variable occurrences at which a variable is not given a new value), our criteria distinguish between two types of uses. P-uses occur in the predicate portion of a decision statement such as while...do, if...then...else, or repeat...until statements. C-uses are all others, including variable occurrences in the right-hand side of an assignment statement, or an output statement.

The intuitive appeal of any of the data flow-based strategies is that by requiring test cases that stress the interactions between parts of a program connected by the flow of data as well as the flow of control, one is likely to uncover faults. In particular, in order for a failure to occur, two things are
necessary. First a fault must be present, and second there must be some way for the effects of the fault to be reflected in the output. Both Morell [13], and Richardson and Thompson [17] discussed this perspective on testing. Most failures involve the execution of an incorrect definition, either because there is an incorrect assignment statement or input statement, or a predicate is faulty, (and hence an incorrect path is taken which leads to an incorrect definition being executed) or a definition is missing, (and hence an incorrect definition has been performed—the null definition). If a variable is assigned a value at some point in a program, and no test set includes a path on which that value is ever used, it is extremely unlikely that an incorrect definition will be detected by testing. By varying the types of uses emphasized, it is expected that different types of faults are likely to be detected by different criteria.

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 their combination. For example, all-definitions requires sufficient test data such that every definition is used at least once. 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 definition. The all-c-uses criterion 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 definition. All-p-uses requires that test data be included which cause the traversal of at least one path from each variable definition to every p-use of that definition. All-uses requires that test data be included which cause the traversal of at least one subpath from each variable definition to every p-use and every c-use of that definition. A definition-use association is a triple consisting of a variable \( v \), a node \( d \) in the program's flow graph in which \( v \) is defined, and a node or edge \( u \) in which \( v \) is used, and for which there is a path from \( d \) to \( u \) on which \( v \) is not redefined. All-du-paths 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 definition. Such a subpath is known as a du-path. Precise definitions of the entire family of criteria are included in [16] for a very simple language, and in [6] for Pascal.

III. DESIGN OF THE STUDIES

In this study we considered routines from Algorithm 1 to Algorithm 200 of the ACM suite that contained known faults, and five or more decision statements. As in our earlier study, we decided that for programs with fewer than five decision statements, the maximum number of test cases needed to satisfy even the most demanding criterion is modest, and there is no difficulty running that number of test cases. We list the numbers of the algorithms from the ACM suite included in this study in the Appendix.

In our earlier study, testers were instructed to select groups of test cases using the selection strategy of their choice, other than a data flow criterion. Each tester's goal was to do a "good" job of testing and to use the data flow criteria only as adequacy criteria. Typically if the criterion was not satisfied after several iterations, the tester selected test cases with the explicit intent of exercising definition-use associations needed to satisfy the chosen criterion. In the present study, a somewhat different selection procedure was followed.

Since we wanted to determine the contribution of each individual test case, tests were processed one at a time for each criterion. Nonetheless, the tester still selected the test cases in groups, independent of the particular data flow adequacy criterion being used. ASSET was used to determine the number of remaining definition-use associations after each test case was run. Once a criterion was satisfied, no additional test cases were run for that criterion.

As in our earlier study, no attempt was 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 that might fulfill several characteristics simultaneously, testers were instructed to select "natural" test cases, each with a single purpose. This was done because testing practitioners frequently do not attempt to minimize test sets and the use of "atomic" test cases yields pessimistic results. Nonetheless, our procedure (running test cases one at a time and stopping as soon as the criterion was satisfied) yielded test sets that were at least close to minimal. Note, however, that another tester could certainly have selected a different set of test cases to satisfy a given criterion, and that test set could have been of substantially different size. We discuss the question of tester variability in Section VII, and present the results of a small study addressing this issue.

As in the earlier study, the issue of unexecutable subpaths was an important problem. If the selected criterion requires the traversal of unexecutable subpaths, then no test set, regardless of how extensive, can satisfy the criterion and thereby adequately test the program. In such a case, we say that a test set almost satisfies a data flow criterion provided it causes every executable subpath required by the given criterion to be traversed. In the sequel, we shall say that a criterion has been satisfied provided that it has been almost satisfied. In Section V, we discuss the issue of unexecutable paths.

In the study which used the ACM suite, one person served as the tester. He had had approximately two years industrial experience prior to this study. In our earlier study using the Kernighan and Plauger suite, there were three testers, including the one who participated in this study. They had professional experience ranging from 2 to 8 years. All were paid a full-time salary to work on the study, and considered it a full-time job for the duration of the study.

IV. COST RESULTS

In this study we collected the same information and computed the same values for the ACM suite as we did for the Kernighan and Plauger suite. This included:

- the least squares line: \( t = od + \beta \), 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
- 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
TABLE I
RESULTS FOR BOTH STUDIES

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>least squares(K&amp;P)</td>
<td>$t = 0.52d + 1.87$</td>
<td>$t = 0.07d + 1.01$</td>
<td>$t = 0.81d + 1.42$</td>
<td>$t = 0.93d + 1.40$</td>
</tr>
<tr>
<td>least squares(ACM)</td>
<td>$t = 0.31d + 2.77$</td>
<td>$t = 0.30d + 2.69$</td>
<td>$t = 0.32d + 3.02$</td>
<td>$t = 0.38d + 5.06$</td>
</tr>
<tr>
<td>weighted average(K&amp;P)</td>
<td>$t = 0.43d$</td>
<td>$t = 0.70d$</td>
<td>$t = 0.72d$</td>
<td>$t = 0.81d$</td>
</tr>
<tr>
<td>weighted average(ACM)</td>
<td>$t = 0.46d$</td>
<td>$t = 0.47d$</td>
<td>$t = 0.48d$</td>
<td>$t = 0.54d$</td>
</tr>
<tr>
<td>maximum $t/d(K&amp;P)$</td>
<td>3.5</td>
<td>2.33</td>
<td>3.67</td>
<td>3.67</td>
</tr>
<tr>
<td>maximum $t/d(ACM)$</td>
<td>1.5</td>
<td>0.83</td>
<td>1.5</td>
<td>2.17</td>
</tr>
</tbody>
</table>

TABLE II
COMBINED RESULTS

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>least squares</td>
<td>$t = 0.36d + 2.82$</td>
<td>$t = 0.38d + 3.17$</td>
<td>$t = 0.39d + 3.76$</td>
<td>$t = 0.49d + 4.03$</td>
</tr>
<tr>
<td>weighted average</td>
<td>$t = 0.44d$</td>
<td>$t = 0.60d$</td>
<td>$t = 0.62d$</td>
<td>$t = 0.69d$</td>
</tr>
</tbody>
</table>

TABLE III
RESULTS FOR BOTH STUDIES

<table>
<thead>
<tr>
<th></th>
<th>K&amp;P</th>
<th>ACM</th>
<th>COMBINED</th>
</tr>
</thead>
<tbody>
<tr>
<td>average all-uses/all-defs</td>
<td>1.79</td>
<td>1.72</td>
<td>1.76</td>
</tr>
<tr>
<td>average all-du-paths/all-defs</td>
<td>1.96</td>
<td>1.97</td>
<td>1.96</td>
</tr>
<tr>
<td>average all-du-paths/all-uses</td>
<td>1.08</td>
<td>1.15</td>
<td>1.10</td>
</tr>
</tbody>
</table>

- 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
- the weighted average of the ratios of the number of test cases sufficient to satisfy all definitions to the number of test cases sufficient to satisfy all uses
- the weighted average of the ratios of all uses to the number of test cases sufficient to satisfy all-du-paths
- the weighted average of the ratios of all-p-uses to the number of test cases sufficient to satisfy all-du-paths.

A discussion of why these values were studied, and the details of how the computations were performed is contained in [18].

The cost portion of the current study used 15 routines selected from the ACM suite, ranging in size from 5 to 16 decision statements. The average number of decision statements was 12. The Kernighan and Plauger study used 29 routines, ranging in size from 5 to 15 decision statements, with the average number being 8.

As Table I indicates, the average costs of satisfying the data flow criteria for the ACM study, using either the least squares analysis or weighted average, were generally even more favorable than they were in the Kernighan and Plauger study. Not only can we expect to need fewer than $d$ test cases for a routine with $d$ decision statements, the predicted value was usually smaller using the ACM suite than using the Kernighan and Plauger suite.

For comparison purposes, Table I includes the results of the first three items from our earlier study as well as the new study; the results for the last three items for both studies are presented in Table III. In Table II we include the results for the first three items for the 44 routines, (29 Kernighan and Plauger routines plus the 15 ACM routines) considered as a single suite. The information for the last three items is included in Table III for the combined suite.

The Kernighan and Plauger study indicated that one can expect that if a program contains $d$ decision statements, the tester will need fewer than $d$ test cases to satisfy any of the data flow criteria. This is shown in rows 1 and 3 of Table I.

The ACM study indicated the same thing. This is shown in rows 2 and 4 of Table I. Rows 5 and 6 of Table I show that the worst case encountered in either study required only 3.67$d$ test cases to satisfy the most demanding criterion, all-du-paths. For the ACM study, the value was 2.17$d$.

Table III shows that for both suites it was relatively inexpensive, on average, to use the all-du-paths criterion as compared to much less demanding criteria such as all-definitions. In all cases, cost was assessed purely in terms of the number of test cases sufficient to satisfy the given criterion. In Section V we discuss the appropriateness of basing this assessment purely on this factor.

Table III shows that the relative costs of satisfying weak criteria versus demanding criteria were almost the same for both studies. In both studies, fewer than twice as many test cases were required, on average, to satisfy all-du-paths as compared to all-definitions. The column labeled COMBINED was based on treating the two suites of programs as a single suite of 44 programs.

V. ADDITIONAL OBSERVATIONS

In the Kernighan and Plauger study, we observed that although the number of test cases needed to satisfy all-du-paths remained modest as the size of the program grew, the number of unexecutable du-paths might become very large, and determining which du-paths were unexecutable sometimes required a substantial amount of effort. (In general it is undecidable whether a given subpath is executable.) Not surprisingly, we observed that same phenomenon in this study.

For a given program, the unexecutability rate for a data flow criterion other than all-du-paths is the ratio of the number of unexecutable definition-use associations to the total number of definition-use associations in the program required by the criterion. For the all-du-paths criterion, the unexecutability rate...
is the ratio of the number of unexecutable du-paths to the total number of du-paths in the program. For two thirds of the programs tested from the ACM suite, more than 50% of the du-paths were determined to be unexecutable. In particular, for the 15 programs in this study, there were on average 182 du-paths per program, 102 of which were determined to be unexecutable, for an average unexecutability rate of 56%.

The implication of large numbers of unexecutable du-paths is that assessing the cost of data flow testing only in terms of the number of test cases needed to satisfy a criterion might yield an overly optimistic picture of the real effort needed to accomplish the testing.

We also considered the number of unexecutable definition-use associations for other data flow criteria when we determined that the relevant criterion had been (almost) satisfied for the given program from the ACM suite. For all-c-uses there were, on average, 71 definition-use associations with 8 unexecutable associations per program, for an average unexecutability rate of 11%. For all-p-uses there were, on average, 72 definition-use associations with 11 unexecutable ones per program, for an average unexecutability rate of 15%.

We thought that perhaps the percentage of unexecutable definition-use associations and particularly the percentage of unexecutable du-paths might be uncharacteristically high since the programs in the ACM suite were frequently highly unstructured. This is not surprising since these programs were written before it was common to worry about programming style and the effects of the unrestricted use of GOTO statements. We therefore calculated the average unexecutability rates for the programs of the Kernighan and Plauger suite. The programs in this suite are highly structured and very carefully written with a particular emphasis on style and clarity. Nonetheless, we found the average unexecutability rate for all-c-uses to be 9%, for all-p-uses 18%, and for all-du-paths 49%. The lack of program structuredness alone, therefore, did not seem to account for the high percentage of unexecutable du-paths found for the ACM suite.

Note that these are just rough guidelines for the tester since there was a substantial amount of variation in the unexecutability rates among programs in each study. The high, low, and average values of these ratios for both studies and each criterion are shown in Table IV.

<table>
<thead>
<tr>
<th>TABLE IV</th>
<th>VARIATION IN UNEXECUTABILITY RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c-uses</td>
</tr>
<tr>
<td></td>
<td>max</td>
</tr>
<tr>
<td>K&amp;P</td>
<td>0.29</td>
</tr>
<tr>
<td>ACM</td>
<td>0.24</td>
</tr>
</tbody>
</table>

unexecutable path segments were frequently easy to detect, the total time required was not negligible. If a path segment is unexecutable, any path including that segment is also unexecutable. Therefore, we wrote a routine to remove from consideration any associations which include segments already shown to be unexecutable. This frequently substantially pared down the number of associations the tester needed to examine manually. Still, it was often a time-consuming and error-prone task to determine unexecutability.

In view of the very high number of unexecutable du-paths encountered when trying to satisfy all-du-paths, we decided to consider the size of test sets sufficient to satisfy the all-definitions and all-edges criteria for programs in the ACM suite. All-edges, also known as branch testing, requires that sufficient test cases be included to ensure that every edge in the flow graph representing the program is traversed at least once. The unexecutability rate for the all-edges criterion is the ratio of the number of unexecutable edges in the flow graph representing the program, to the total number of edges in the flow graph representing the program. The all-definitions criterion is described in Section II.

For all-definitions, the least squares analysis produced the following relationship between the number of decision statements and the required number of test cases:

\[ t = 0.16d + 1.90. \]

For all-edges, the relationship was

\[ t = 0.25d + 1.53. \]

When the weighted average analysis was used, we found the relationship for all-definitions to be

\[ t = 0.20d \]

and for all-edges:

\[ t = 0.22d. \]

Table V summarizes the number of test cases sufficient to satisfy the all-definitions and all-edges criteria indicated by the least squares analysis for routines containing 5, 12, and 16 decision statements. For comparison, results for the other data flow criteria are included.

Table VI summarizes this data for the weighted average analysis. Again results for the other data flow criteria are included for comparison purposes. In all cases, results were always rounded up. Thus, an average value of 2.01 would be shown in Tables V or VI as 3.

These results indicate that all-definitions required, on average, slightly fewer test cases to satisfy than all-edges, and both required fewer test cases to satisfy than the data flow criteria. In all cases the tester could expect to need a very
modest number of test cases to satisfy any of the criteria. But, as mentioned earlier, the big “hidden” cost involves determining whether definition-use associations or du-paths are unexecutable. As we noted earlier, for the ACM suite, the average unexecutability rates were 11% for the all-c-uses criterion, 15% for the all-p-uses criterion, and 56% for the all-du-paths criterion. In contrast, all-definitions and all-edges both had unexecutability rates of 0.

VI. FAULT DETECTION RESULTS

We had intended to use the ACM suite to study the fault detection capabilities of the data flow criteria, and in fact such a study was performed. We had selected the ACM suite of programs since it has several important characteristics: the programs were written to accomplish necessary tasks (as opposed to being written for the experiment), they were written by professional programmers (as opposed to students or novices), and they contained real (as opposed to seeded) faults. In addition, the format of the ACM suite was particularly attractive for such a study. For each program in the suite, a brief specification is included, along with the contributed program. If faults were subsequently found in the program, these faults were identified and corrections proposed, generally by users other than the original programmer. This information, too, was published with the original program. Therefore, by looking through the documentation, we were able to identify which programs had known faults without looking to see what the faults were. This was essential to ensure that we did not bias our selection of test data by having knowledge of fault location and type.

In this portion of the study, we examined 31 routines containing 47 documented faults. All of the 15 routines used in the cost portion of the study were included among the 31. Of the 47 faults, 33 were detected by a test set satisfying at least one of the data flow criteria. In addition, one fault not mentioned in the documentation was detected. Of the total of 34 faults exposed, 32 were exposed by test sets which satisfied all of the considered data flow criteria (all-c-uses, all-p-uses, all-uses and all-du-paths). One fault was exposed only by a test set satisfying all-du-paths, and one fault was exposed by the test set which satisfied all-p-uses and all-uses, and by the test set used to satisfy all-du-paths, but not by the test set satisfying all-c-uses. The remaining 14 faults went unexposed by any test set used to satisfy one of the data flow criteria. Therefore, 71% of the known faults were exposed by at least all-du-paths, and 67% were exposed by all-c-uses, all-p-uses, all-uses, and all-du-paths.

Of the 14 faults not exposed by any test case, 13 were special value faults. That is, a failure only occurred for one or a very small set of special cases. Typically there was nothing in the specification that singled these values out for special treatment. Thus they were very unlikely to be detected by either black-box or white-box testing techniques.

Similar results have been reported by Nafos [14]. In his studies involving required pairs testing, a different form of data flow-based testing, he found that the criterion was poor at detecting both special value and boundary errors. In [15] he noted that none of the structural testing strategies was particularly effective at detecting these types of faults.

This is confirmed, too, in a recent paper [4] in which Foreman and Zweben considered the effectiveness of control and data flow testing strategies. They concluded that:

The results of this analysis give empirical evidence to support the conclusion that control and data flow strategies are effective for finding program defects. ¹ They further concluded that:

when control and data flow strategies are not sufficient, often another well-known technique such as special value testing, boundary testing, static data flow analysis, or some generalized strategy is successful. ²

After completing the fault detection portion of the study and examining the results, we concluded that certain characteristics of the programs from the ACM suite made them inappropriate candidates for this study. Since we had consciously avoided looking at the faults reported in the programs in order not to bias our results (it is certainly easier to find faults when you know what they are and where they are located) we were not aware of this problem until the conclusion of the study.

In particular, we found that some of the detected faults were probably not introduced during coding, but rather during some aspect of the publication process. Since these programs were written prior to the widespread use of electronic processing, most of the programs were probably run using punch cards. When submitting the program for publication, a copy was probably typed up and mailed. In particular, in some cases it is unlikely that the version that was actually run, was the same as the version that was submitted for publication in CACM. Several of the faults, therefore, were probably just typographical errors introduced either in typing or when the program was typeset for publication. We believe this to be true because in some of these cases virtually any test case would have exposed the fault. It is therefore difficult to imagine that the program could have ever been run in the published form. We therefore do not consider the results of this portion of the study indicative since we are unable to provide reliable information on the fault detection rate, nor do we really know

¹ page 221.
² page 222.
which of the faults were true programming faults, and which were artifacts of the publishing process.

VII. TESTER VARIABILITY

Lauterbach and Randall [12] performed a study to compare the cost and fault detection abilities of several different testing techniques. One conclusion reached in this paper was that the choice of tester was seen to be a larger factor in the resulting effectiveness of the testing than was choice of test technique. Since we had found very positive results in terms of the number of test cases sufficient to test programs using the data flow criteria for both the string processing and numerical programs, we wondered to what extent our results were a function of our testers' talents.

In particular, we wondered how much variation we would find among different testers. We therefore selected four routines used in the Kernighan and Plauger study and required all the students in a graduate course on software testing to test each of the routines using each of the four data flow criteria evaluated in the earlier studies. Several students in the class work full-time while attending our graduate program part-time. When submitting this assignment, students were asked to describe briefly all of their professional experience along with the results of the tests. The only special training the students received prior to beginning the assignment was a one hour lecture describing the basic intuition and definitions of data flow testing, and a description of each of the criteria. The teaching assistant gave a brief demonstration of ASSET, distributed the user's manual, and was available to answer students' initial questions.

The goal of this very small, informal study was to see the amount of variation in the number of test cases used to satisfy each of the data flow criteria. Nine students successfully completed the assignment. Of these, five had 0 to 6 months of industrial experience, and four had 4 to 8 years experience. For the purposes of this study, we only considered the four students with substantial experience since we felt the results for the others would not be indicative of professional programmers or testers. The person who had tested the four selected routines in the original study had had eight years of industrial experience prior to the study. Tables VII–X show the average number of test cases used by each tester for each of the criteria all-c-uses, all-p-uses, all-uses, and all-du-pairs, for the four testers for the four routines considered in this study. The table heading indicates the name of the routine from [10]. The last row of each table shows the results for the program obtained in the original study, i.e. the result that contributed to the least squares and weighted average analysis for the Kernighan and Plauger suite.

Since we considered four programs and four criteria in this study, there should have been a total of 16 values for each tester. For AMATCH, tester 3 only completed testing using the all-c-uses criterion, while the tester in the original study did not complete the all-du-paths criterion for SUBLINE. Of the 15 average values for the number of test cases computed

\[\text{TABLE VII}\]

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>tester 1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>tester 2</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>tester 3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>tester 4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>4.5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>original</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

\[\text{TABLE VIII}\]

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>tester 1</td>
<td>24</td>
<td>15</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>tester 2</td>
<td>16</td>
<td>13</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>tester 3</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>tester 4</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>average</td>
<td>17.25</td>
<td>15</td>
<td>17.5</td>
<td>17.5</td>
</tr>
<tr>
<td>original</td>
<td>21</td>
<td>14</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

\[\text{TABLE IX}\]

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>tester 1</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>tester 2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>tester 3</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>tester 4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>average</td>
<td>6.5</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
</tr>
<tr>
<td>original</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

\[\text{TABLE X}\]

<table>
<thead>
<tr>
<th></th>
<th>all-c-uses</th>
<th>all-p-uses</th>
<th>all-uses</th>
<th>all-du-paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>tester 1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>tester 2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>tester 3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>tester 4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>average</td>
<td>2.25</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>original</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>

in this study compared with the values found in the original Kernighan and Plauger study, nine were lower than the number found in the original study. Of the other six values, all were within 1.5 of the original. This indicates that the original results were not overly optimistic.

We were also curious whether the amount of professional experience directly affected the results. Of the four testers who participated in this study, two had eight years experience each, while the other two testers had four years of experience. In each of the 16 cases, the average number of test cases used to satisfy a criterion for a program was lower for the two testers with four years experience than the two with eight years experience. Evidently once a programmer was fairly experienced, the primary factor determining how efficiently they selected test cases was personal skill rather than the amount of experience.

\[\text{page 40}\]
VIII. SUMMARY AND CONCLUSIONS

We observed a number of things about data flow testing in these small, informal studies. Using the ACM suite, we confirmed our earlier observation that it is relatively cheap and easy to satisfy the data flow testing criteria, when the cost was assessed purely on the basis of the number of test cases sufficient to (almost) satisfy the criteria. Also, there was little difference in terms of the number of test cases sufficient to satisfy the least demanding criterion considered, all-definitions, and the most demanding criterion, all-du-paths. This is similar to the results found in our earlier study using the Kernighan and Plauger suite [18].

However, we acknowledge that there was a hidden cost of satisfying some of these criteria, particularly all-du-paths. Although it required, on average, less than twice as many test cases to satisfy all-du-paths as it did to satisfy all-definitions, it was substantially more difficult to determine whether or not all-du-paths had actually been satisfied. For the ACM suite, in the case of all-definitions, for two thirds of the programs in the suite, there were no executed definition-use associations remaining when the criterion was determined to be satisfied. In contrast to this, in two thirds of the programs used in the cost portion of the ACM study, over 50% of the du-paths were determined to be unexecutable, and it was frequently a difficult and time-consuming job to determine which du-paths were truly unexecutable.

The same issue of hidden costs should be considered for all program-based testing criteria. Generally the issue involves determining unexecutable paths. In the case of mutation analysis [3], a program-based testing strategy that requires that test cases be produced that distinguish the given program from syntactically similar inequivalent programs, the tester must determine which programs are actually inequivalent. This is again an undecidable problem, and equally difficult for humans to determine. We typically do not factor this in when empirically assessing the cost of using such a criterion, but our experience leads us to conclude that we should consider this to be an important factor.

We had selected the ACM suite of programs since it has several important characteristics: the programs were written to accomplish necessary tasks, they were written by professional programmers, and they contained real faults. We feel that these characteristics are very important for conducting meaningful empirical studies that can increase our experience with various testing criteria. However, although the programs were satisfactory for the cost portion of the study, for reasons mentioned above, they were not appropriate for studying the fault detection ability.

It is important that we gain experience with the cost and effectiveness of using different testing strategies, and several attempts have been made to determine that empirically in recent years. There have also been recent papers comparing the effectiveness of testing strategies analytically. These include [19], [7], and [8]. We hope to see an increasing number of papers addressing these issues both empirically and analytically. The introduction of new testing techniques is certainly an important area of software engineering research, but without proper assessment, the value of these new strategies is dubious.

APPENDIX

ACM ALGORITHMS USED IN THE STUDY

* 19
* 21
* 23
* 28
* 34
* 40
* 41
* 42
* 47
* 48
* 49
* 50
* 55
* 56
* 58
* 62
* 65
* 85
* 105
* 106
* 121
* 123
* 124
* 140
* 147
* 149
* 158
* 163
* 180
* 181
* 195

* denotes an algorithm used in the cost portion of the study.

ACKNOWLEDGMENT

I am very pleased to thank B. Jeng who performed most of the testing described in this study. The original version of ASSET was written primarily by P. Frankl and S. Weiss. Additions and modifications were made by E. Campbell and B. Jeng. These studies could not have been performed without their work.

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 a Professor of Computer Science at the Courant Institute of Mathematical Sciences, New York University, NY, and a Member of the Technical Staff of the Software and Systems Research Center, AT&T Bell Laboratories, Murray Hill, NJ. Prior to that, she was on the faculty of the City University of New York, NY, and was a systems engineer for IBM. Her research interests are in software engineering, particularly software testing and reliability and software complexity measures and has published many papers in those areas. She is also interested in the theory of computation and authored (with Martin Davis) *Computability, Complexity, and Languages* (New York: Academic).

Dr. Weyuker is a member of the Executive Committee of the IEEE Computer Society Technical Committee on Software Engineering and the editorial board of the Association for Computing Machinery Transactions on Software Engineering and Methodology (TOSEM). She is also a member of the ACM Committee on the Status of Women and the ACM SIGSOFT and has been an ACM National Lecturer.