Module Testing Techniques
- which are the most effective?

Results of a Retrospective Analysis

Stuart C. Reid
Cranfield University
Royal Military College of Science,
Shrivenham, Swindon, Wilts SN6 8LA, UK

Phone: +44 (0)1793 785490, Fax: +44 (0)1793 783192
Email: reids@rmcs.cranfield.ac.uk

Abstract

If you are module testing as part of software development then you face a predicament: Which techniques, do you use to design the test cases, and which measures of test coverage do you require to be achieved? Industry guidelines and standards often mandate the required testing and many developers believe they know the appropriate level of testing for their system. However, what is the basis of this knowledge? This paper argues that we are currently basing our testing decisions on unsafe foundations and presents results from an empirical study, which appear to refute common testing wisdom.

A retrospective analysis of an operational system was performed to determine what would have been the effectiveness of the most commonly-used testing techniques if applied to an operational avionics system of approximately 20,000 lines of Ada code, produced to satisfy an 'essential' level of criticality.

The test case design techniques covered were equivalence partitioning, boundary value analysis, branch, branch condition, branch condition combination, modified condition decision coverage and random testing; all the techniques were applied as defined in the BCS Software Component Testing Standard.

The results of the analysis show, for this system, that several widely-held assumptions, some apparently supported by industry standards, do not hold. For instance, testing to achieve 100% branch coverage would have detected fewer faults than using equivalence partitioning. Also, random testing would have been surprisingly effective. Random testing would have achieved similar levels of effectiveness for the same number of test cases as any of the other techniques studied except for boundary value analysis, and with only six random test cases per module would have outperformed branch, branch condition, branch condition combination or modified condition decision coverage (the level required for safety-critical avionics software). In contrast, boundary value analysis fared particularly well in this study, and over 50,000 random test cases per module would have been required to equal its test effectiveness.

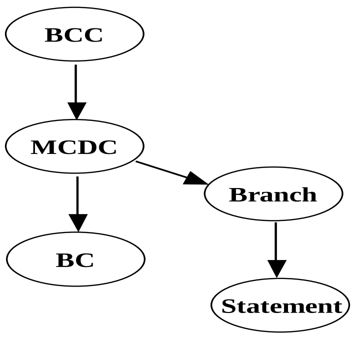
1 Introduction

Many National, International, and Industry Standards mandate the use of specific testing techniques and test adequacy criteria, but what is the rationale for their choice of techniques? In many cases the choice seems arbitrary - in some even contradictory - and even when an informed approach is attempted there is a lack of solid evidence on which to base the decision.

Research on the fault-finding effectiveness of testing techniques has been ongoing for more than twenty years, yet, despite this, there is still no consensus on the most effective techniques. Much of the work has been theoretical, such as providing partial orderings of techniques into subsumes hierarchies or in support of little-used techniques, which, although valuable, provides little help to the practising software developer. Theoretical studies must be supported by results from experiments and case studies. For instance, is there a sound theoretical basis for achieving 85% branch coverage or exercising all boundary values? If not, is there widespread evidence from empirical studies on which we can base our choice? Sadly, the answer appears to be no to both of these questions. Several papers have advocated empirical studies to support the claims made in theoretical studies (or perhaps to suggest new theories), such as [1], [2], [3], [4], and [5]; this paper reports on such a study.

There are, however, problems associated with experiments and case studies. In experiments many factors need to be controlled and measured for the results to be useful. The major difficulty is ensuring that experimental conditions are representative of the real world, so making the results applicable outside the laboratory. For instance the testers should be similar (in training, expertise, enthusiasm, etc.) to those who will use the techniques in industry. Even more problematic is finding real faults to experiment on, as faults artificially seeded into code cannot be truly representative. Case studies pose slightly different problems. The main one being that they are usually carried out after the event, and so useful data is often lost. One way around several of these problems is retrospective analysis, which uses a combination of the experiment and the case study.

The hypotheses that formed the basis for the study were that both Boundary Value Analysis (BVA) and branch testing were more effective than Equivalence Partitioning (EP), that the subsumes ordering [6] of the white box techniques considered reflected their relative test effectiveness (see figure 1), and that random testing was less effective than any of the other techniques studied. These hypotheses are described in more detail in section 3.1.



BC - Branch condition

BCC - Branch condition combination

MCDC - Modified Condition Decision Coverage

An arrow indicates the subsumes ordering between techniques, which, in this context, represents the hypothesis that the technique at the start of the arrow is more effective than the technique at which it is pointed.

Figure 1: Partial Subsumes Ordering

The next section provides a brief background to the study. Section 3 describes the methodology used in the study, while sections 4 and 5 present the results and the techniques used to determine their statistical significance. Section 6 discusses the results and, based on them, provides guidance on the application of the techniques studied. Finally, conclusions and references are presented in sections 7 and 8.

2 Background

2.1 The requirement for module testing

It is recognised best practice in software development to detect and remove faults from the software at the earliest opportunity. This is due to the increasing costs of correction later in the life cycle, and supports the recognised quality goal of minimising rework. Data from industry show the detection and correction of faults by module testing to be half as costly as in the next (integration testing) phase [9].

Apart from financial benefits there are several technical reasons for performing module testing:

- module test sets are available for regression testing;
- reusable modules require module test specifications;
- it provides lower complexity testing and easier debugging than in later testing phases;
- project management benefits by introducing the possibility of parallelism into the testing process.

2.2 Test case design techniques and test coverage measures

The definitions of the testing techniques used in this study were taken from the BCS Software Component Testing Standard [6]. This standard includes definitions of both test case design techniques and test measures; for the rest of this paper these will simply be referred to as testing techniques as it is assumed that the test case design technique is applied to achieve 100% coverage of the corresponding test measure. It was assumed that when applying a test techniques the *minimum* number of test cases would be used to achieve full coverage.

In common with most module testing, no operational profile of inputs was available for the modules studied and so a uniform distribution was used for the random generation of inputs.

2.3 Related work

A number of papers have been published in this area, covering both theoretical ([3], [4], [10], [5], [11], [12], [22], [23]), and empirical studies ([13], [14], [7], [8], [15], [16], [2], [17], [24]). Perhaps one of the surprising features of these studies is that the empirical studies tend to pre-date the theoretical work. There is still not enough data available to validate much of the theoretical work and there is little consensus on which testing techniques are the most effective from the empirical studies.

Although there have been many studies on the effectiveness of *generic* partition testing techniques as a class of test techniques, there has been little study of *specific* partition testing techniques, such as EP, BVA and branch testing. Random testing has fared much better, perhaps because it is much easier to generate test inputs. The results of [14], that random testing can be cost effective for many programs, subsequently validated by [5], have also generated interest in this technique, as intuitively it should be far less effective than systematic techniques.

Future empirical studies must provide sufficient details to enable their results to be compared and used to build up a cohesive body of knowledge in the area. A parametric framework has been suggested [1], which would allow experiments to be classified and so both compared and repeated.

2.4 Problems with test effectiveness experiments

Test effectiveness cannot be measured directly, instead experimenters use surrogate measures, known as response variables, such as the number of defects detected, or, in this case, the probability of detecting a fault. A number of other factors that affect the experimental results, but are not measured, are known as state variables, which characterise the experiment. The results should be applicable elsewhere *given the state variables are similar*. Attempting to manage this 'correspondence' does, however, raise a number of questions:

- are the experimental testers working at the same skill level as real testers?
- if less than a statistically large enough number of test cases are chosen from a subdomain in an experiment are they representative of all values in it?
- can seeded faults be used, or are only real faults worthy of study?
- are the experimental module specifications of typical quality and notation?
- are experiments implementation language specific or can results be applied to other languages?
- are the experimental test techniques well-defined and repeatable?

Only a few of the numerous factors that need to be considered in experiments of this kind have been presented here. The full set of state variables that characterise the experiment described in this paper are provided in Appendix A, and reasons for their choice are given in the next section.

3 Experimental methodology

This section initially presents the hypotheses that the study intended to assert, and then describes the experiment and the theory behind it. The experiment comprised five stages: Failure Analysis, Creation of Fault Finding Sets, Creation of Test Case Sets, Derivation of Probabilities, and finally Analysis of Results. All but the last stage, which is described later, are described in this section, and are shown in figure 2.

In this research the fault-finding sets were initially derived (see 3.3) from the faults identified by a failure analysis of the system (see 3.2). The subdomains required to satisfy test coverage criteria were then identified and defined as the test case sets (see 3.4). Both the fault-finding sets and test case sets were defined using techniques borrowed from symbolic execution. Then, using the concepts underlying domain testing, a definitive probability of detection is calculated in terms of 'overlapping' N-space convex polyhedra, which correspond to the test case sets and the fault-finding sets respectively (see 3.5).

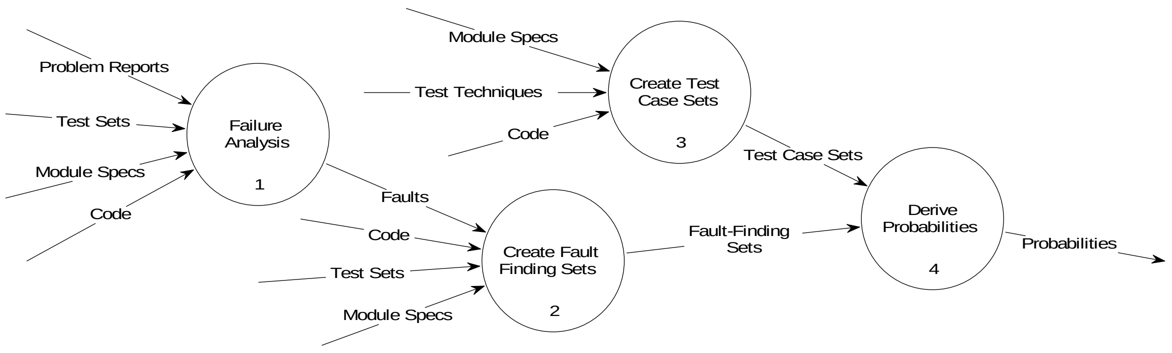


Figure 2: Experimental Methodology

3.1 Hypotheses

The experiment examined thirteen pairwise comparisons. In each case the hypothesis asserted was that the first technique in each pair was more effective than the second technique. These were:

Boundary Value Analysis	-	Equivalence Partitioning
Branch	-	Equivalence Partitioning
Branch	-	Statement
MCDC	-	Branch
MCDC	-	BC
BCC	-	MCDC

plus each of the seven systematic techniques mentioned (above) were hypothesised as being more effective than random testing. For random testing, this was based on using the same number of test cases as required by the technique it was being compared against. For instance, if EP for a particular module required twelve test cases then the probability of detection for the random testing of that module was based on twelve randomly-generated test cases.

3.2 Failure analysis

A project was chosen where the system was already in operation, and a failure analysis was used to identify the faults in the system that could have been detected by module testing. Problem reports were the main source of failure information, as they were required to initiate any change to the system. The reported faults were analysed using the original and modified source code, design specifications and test specifications.

The system has now been flying for some time and the lack of change requests from the customer suggested that most of the faults (that would cause failures) had been documented, and so were included in the failure analysis. The complete system was analysed, which meant that all relevant faults were identified. By this means no experimental bias was introduced by choosing a subset, and it was hoped that a typical and complete cross-section of faults were considered.

Faults were identified from the problem reports, which had to be raised for any change to the code or documentation *after* it was lodged with configuration control. This meant that the faults identified were those that were left after the developers had carried out their informal module testing, which, for the case study project, was branch testing. So any faults detected by this branch testing were not included in the analysis. An average branch coverage of 99.0% was achieved by the developers, the last 1.0% being due to failures in test execution, rather than unreachable or missed code. This 'loss' of data on faults already found by informal testing was not considered a handicap as the faults that were *left* were considered the most important. Hetzel [18] states that the "defects of greatest interest are those still in the product after release, not those already discovered."

The faults that could have been detected by module testing were a small subset of all faults found with the system (less than 10%). They were then used to create the fault-finding sets. The full results from the failure analysis of the system can be found in [19].

3.3 Creation of fault finding sets

Once a fault was identified and understood, the set of inputs to the module that would cause the fault to be exposed were determined and these were used to create fault-finding sets (there will normally only be a single set corresponding to a given fault, however in some cases disjoint sets may expose a single fault). An example is shown in Figure 3, with two fault-finding sets shown.

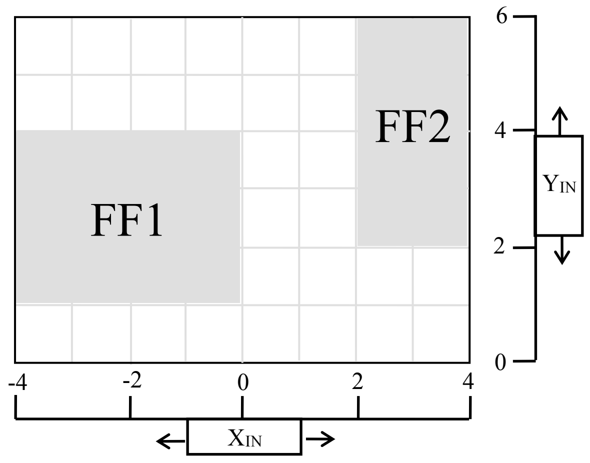


Figure 3: Fault-Finding Sets

These fault-finding sets take the form of simultaneous equalities and inequalities on the module inputs representing the input conditions for exposing the fault. These fault-finding sets bear a strong resemblance to the system of equalities and inequalities used in symbolic execution systems. In the example the fault-finding sets (FF1 and FF2) would be represented by:

$-4 \le X_{IN} \le 0$ <u>AND</u> $1 \le Y_{IN} \le 4$ FF1

$2 \le X_{IN} \le 4$ <u>AND</u> $2 \le Y_{IN} \le 6$ FF2

where the two input parameters are X_{IN} and Y_{IN}, as shown in figure 3.

3.4 Creation of test case sets

The test case sets were all derived by the same person from the definitions of the testing techniques described in section 2, with the aim of minimising the effect of the tester skill level on the experiment.

The test case design techniques considered correspond to test coverage criteria in that ‘complete’ use of the test case design technique will generate a test case suite that will achieve 100% coverage of the test coverage criterion. Test case sets were constructed that corresponded to those input conditions that had to be satisfied to achieve the given test coverage criterion.

As with the fault-finding sets, the test case sets take the form of simultaneous equalities and inequalities on the module inputs. For a given test coverage criterion there will be a number of test case sets, with each set corresponding to one of the subdomains of the module’s input domain required to be exercised by the criterion. For instance, the path coverage criterion for a typical module would lead to a large number of subdomains - one for each of the distinct paths through the module.

For the example, assume that an imaginary test technique requires that four test case sets be satisfied, as shown in figure 4. It can be seen that this imaginary test technique has generated overlapping subdomains (in the corners). This is typical of the practical application of real test techniques where it is unlikely that the partitions will be truly distinct as is generally assumed in theoretical studies of techniques.

3.5 Derivation of probabilities

The fault-finding capability that would be achieved by satisfying the given test coverage criterion is derived by measuring the degree of intersection between the fault-finding sets and the test case sets. The domains of both the fault-finding sets and the test case sets can be considered to be convex polyhedra in N-space, where N is the number of input parameters to the module. The probability of a fault being detected is measured by the ‘overlap’ between the fault-finding polyhedra and the test case polyhedra. In general the probability of detection can be calculated as follows:

Let D denote the module's input domain, of size d, (thus d=|D|), m points of which cause the module to fail. Then these m points shall define the extent of the fault-finding sets, and if FF represents the union of all fault-finding sets, then m=|FF|.

If P_{DET} is the probability that a randomly-chosen input, based on a uniform distribution, will cause a failure (and so detect a fault) then $P_{DET} = {}^{m}/_{d}$.

If TC represents a test case set then the probability of the fault being detected by a single randomly-chosen test case from TC is given by $P_{DET}(TC, FF) = {}^{m_{TC}}/_{d_{TC}}$ where d_{TC}=|TC| and m_{TC}=|FF∧TC|.

Thus, the probability of missing the fault is $P_{MISS}(TC, FF) = 1 - P_{DET}(TC, FF) = 1 - {}^{m_{TC}}/_{d_{TC}}$.

Then, given that n test case sets (TC1,TC2,..,TCn) are required to satisfy the test coverage criterion, C, the probability of the fault (represented by the fault-finding sets FF) being missed is the product of the miss probabilities for each of the n test case sets, thus $P_{MISS}(C, FF) = \prod_{i=1}^{n} P_{MISS}(TCi, FF)$.

And so, the probability of detecting the fault is $P_{DET}(C, FF) = 1 - P_{MISS}(C, FF) = 1 - \prod_{i=1}^{n} (1 - P_{DET}(TCi, FF))$.

For random testing, the input domain (D) for the module can be considered as a single test case set, but, as stated earlier in section 3.1, different numbers of random test cases were used to provide 'fair' comparisons with the other techniques.

Thus the earlier formulae, which provide probabilities based on a single test case per subdomain (test cases set) can be modified for random testing by considering D as a single test case set, with n test cases. Thus, $P_{DET}(RANDOM, FF) = 1 - (1 - P_{DET}(D, FF))^{n}$.

This assumes that test cases are selected with replacement. As the number of test cases is very small when compared with the domain sizes this should have minimal effect, and if random inputs were generated automatically this would probably be the case anyway.

An example calculation can be shown by considering the fault-finding sets (FF1 and FF2) and the test case sets (TC1 to TC4) described earlier in sections 3.3 and 3.4, respectively, and shown in figures 3 and 4.

$-4 \le X_{IN} \le -3$ AND $0 < Y_{IN} \le 6$	TC1
$-4 \le X_{IN} \le 4$ AND $5 \le Y_{IN} \le 6$	TC2
$3 \le X_{IN} \le 4$ AND $0 \le Y_{IN} < 6$	TC3
$-4 < X_{IN} < 4$ AND $0 \le Y_{IN} \le 1$	TC4

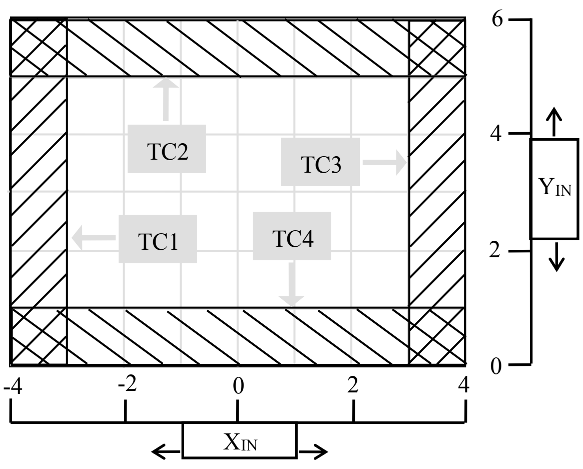


Figure 4: Test Case Sets

From a consideration of the overlap of the fault-finding sets and the test case sets, the probabilities of detection can be determined for each of the four test case sets, thus:

$$P_{DET(TC1)} = 3/6 \Rightarrow P_{MISS(TC1)} = 3/6 \qquad P_{DET(TC2)} = 2/8 \Rightarrow P_{MISS(TC2)} = 6/8$$

$$P_{DET(TC3)} = 4/6 \Rightarrow P_{MISS(TC3)} = 2/6 \qquad P_{DET(TC4)} = 0 \Rightarrow P_{MISS(TC4)} = 1$$

Given that one test case is used per test case set then the probability of all four missing the fault is the product of the probability of missing for each of the individual test case sets:

$P_{MISS} = 3/6 * 6/8 * 2/6 * 1 = 36/288 = 0.125$, or 12.5%.

So, the overall probability of detection, P_{DET} = (1 - 0.125) = 0.875, or 87.5%.

4 Results

Table I contains the probabilities of detecting the faults for each of the different testing techniques considered for each of the 17 modules. Each module contained a single fault, identified in the failure analysis. Where a value of one appears in the table then if the specified test technique had been applied to the given module it would have been certain to detect the fault. Conversely, values of zero imply that application of the test technique would never have discovered the fault. All intermediate values represent the probability of detecting the known fault.

Branch testing had been carried out by the developers before the study began, and 100% branch coverage had been achieved for each of the 17 modules. This did not, however, mean that all faults that could have been found by branch testing had already been detected, as can be seen from Table I, where there are no zero probabilities of detection for any of the faults for branch testing. The probabilities of detection calculated for branch testing raised a question in that they contained three module/faults (ID=10, 11 and 17) with a probability of detection of one - as 100% branch coverage had already been achieved, then why had these faults not been discovered? After investigation it was found that these faults had been hidden from the original branch testing by errors in variable definitions in the test environment (test harness, stubs, etc.). For this reason, and that all the test techniques apart from random testing would have discovered the fault, it was decided that the results from these three modules would be discounted from the comparison of techniques and the statistical analysis.

Table II contains the collated results for the remaining 14 modules. The 'mean' value in the table can be interpreted as the average probability of detecting a fault in a module (in this sample) by the use of the given technique. The 'sum' can be interpreted as the average number of faults that would have been found across the 14 modules by the use of the given technique. The '#cases' value is the average minimum number of test cases required to achieve 100% coverage by the technique. The final row, 'Random Equiv' provides the average test effectiveness for random testing if applied using the same number of test cases per module as the systematic technique.

5. Statistical analysis

Statistical measures were applied to the thirteen pairs of testing techniques (as described in section 3.1) using the probability of detection of faults as the measure of test effectiveness.

Hypothesis testing was used to determine whether the different techniques provided any significant increase in test effectiveness. The null hypothesis in each case was that there was no real difference in the effectiveness of the two techniques, while the alternate hypothesis was that one technique was significantly better.

Two separate statistical tests were used to assert the null hypothesis, due to the unknown distribution of the fault detecting probabilities. Initially a technique based on a normal distribution was used (the correlated form of Student's t-test [20]), the results of which can be seen in Table III, labelled 't'. As tests for population means are generally insensitive to departures from normality, this was considered reasonable, however a second set of tests were performed, that are applicable whether the distribution is normal or not, to confirm the results. The Mann-Whitney form of Wilcoxon's test with correlated samples [20] was used, the results of which can also be seen in Table III, labelled 'U'.

TABLE I: Probabilities of Detection

Module ID	EP	BVA	Statement	Branch	BC	MCDC	BCC	Random
1	0.000055	0.72	0.0000714	0.0000714	0.0000714	0.0000714	0.0001197	0.00001710425
2	0.00017413	0.72	0.0000684	0.0000684	0.0000684	0.0000684	0.0000684	0.0000342085
3	0.0001085	0	0.0000342085	0.0000342085	0.0000342085	0.0000342085	0.0000342085	0.0000342085
4	0.0000215	0	0.0000915	0.0000915	0.0000915	0.0000915	0.0001197	0.00001534
5	0.00001529	1	0.0000305	0.0000305	0.0000305	0.0000305	0.0000305	0.00000001017
6	0.09375	0	0.09375	0.09375	0.1362	0.1787	0.2557	0.03515625
7	0.578125	1	0.613	0.76	0.615	0.851	0.851	0.3804065
8	0.85086	1	0.000348	0.000348	0.000348	0.000348	0.000348	0.0000868055
9	0.00052074	1	0.0001786	0.0001786	0.0001786	0.0001786	0.0001786	0.0000868055
10	1	1	1	1	1	1	1	0.397
11	1	1	1	1	1	1	1	0.401
12	0.00006499895	1	0.001072	0.001072	0.001072	0.001072	0.001072	0.000015259
13	0.00003322	1	0.00003322	0.00003322	0.00003322	0.00003322	0.00003322	0.0000076294
14	0.00052074	1	0.4375	0.4375	0.4375	0.4375	0.4375	0.125
15	0.0276	0.9961	0.0276	0.0276	0.0276	0.0276	0.0276	0.0060763888
16	0.75	0.9961	0.75	0.75	0.75	0.75	0.75	0.21875
17	1	1	1	1	1	1	1	0.401

TABLE II: Probabilities of Detection- Results for Reduced Set (14 modules)

	EP	BVA	Statement	Branch	BC	MCDC	BCC	Random
mean:	0.16	0.75	0.14	0.15	0.14	0.16	0.17	0.12
sum:	2.3	10.4	1.9	2.1	2.0	2.2	2.3	0.8
#cases:	5.1	13.7	3.5	4.2	4.1	4.4	4.7	1
Random Equiv:	0.16	0.20	0.12	0.15	0.14	0.16	0.16	n/a

Table III : Statistical Measures - Results for Reduced Set (14 modules)

	Branch-EP	BVA-EP	Branch-State'nt	MCDC-Branch	MCDC-BC	BCC-MCDC	EP-Random	BVA-Random	State'nt-Random	Branch-Random	BC-Random	MCDC-Random	BCC-Random
P_R (t)	insig.	<<0.005	insig.	insig.	insig.	insig.	insig.	<<0.005	insig.	insig.	insig.	insig.	insig.
P_R (U)	insig.	<0.01	insig.	insig.	insig.	insig.	insig.	0.0175	insig.	insig.	insig.	insig.	insig.

Both of the statistical tests were applied using their correlated form, which is applicable when there is correspondence between the i_{th} member of each sample under test. This meant that the probabilities of detection were considered in pairs, with the specific faults and modules as the connecting factor. This correlation technique allows the specific effects that are common to the pair to be minimised.

In each of the comparisons the value P_R was the risk of being wrong if the null hypothesis was rejected. Thus, ideally, the values calculated for P_R should have been as low as possible, so that the null hypothesis could be rejected in favour of the alternate hypothesis (that one technique *was* more effective than the other). Where the test found no significant difference between techniques then 'insig.' was entered. This corresponds to a value greater than 0.05 for $P_R(t)$ and greater than 0.1 for $P_R(U)$.

6. Discussion

6.1 Branch testing and EP

When the effectiveness of branch testing and EP were compared the statistical measures showed no significant difference between them. Branch testing had been expected to perform better, but, surprisingly, EP had a marginally higher mean test effectiveness than branch testing. It was thought that this was partly due to the module specifications being written in fairly low-level structured english thereby providing EP with a level of detail similar to that of the code. EP also required slightly more test cases, but this was probably due to the deliberate optimisation of logic performed during the generation of the code from the module specifications.

As the system had already been informally tested to achieve 100% branch coverage then several of the faults that could have been expected to be found by branch testing (and EP) had been removed already, perhaps contributing to the relatively poor performance of these techniques when compared with others not based on control flow, such as BVA.

6.2 BVA and EP

The mean probability of detection for BVA was more than four times that for EP and both statistical techniques supported the hypothesis that BVA was more effective than EP. The cost, however, of this higher effectiveness was that more than twice as many test cases were required. Despite the extra test cases BVA appears far more attractive to the practitioner. On average, eight extra faults would have been found by using BVA (out of a possible total of 14) and the difference in the number of required test cases does not reflect a proportionate increase in the cost of applying BVA over EP. The major cost of applying *either* technique is in identifying the equivalence partitions (from which the boundaries are derived for BVA). The extra cost introduced by using BVA is mainly in the generation of the expected results for the additional test cases, as, by using automated test tools, the execution and checking of these extra tests will not incur any significant costs.

Despite the results appearing to put BVA far ahead of the other techniques in term of overall test effectiveness, BVA was the only technique for which zero values for probabilities of detection were recorded. This is because BVA targets boundaries, occasionally to the exclusion of other parts of the input domain that contain faults. This suggests that even better results could be achieved by using BVA in concert with a second technique that was known to 'cover' the gaps it leaves in the input domain. Further research is required to determine which other technique fulfils this role most effectively.

6.3 Branch and Statement

When comparing branch and statement testing the statistical measures highlighted no significant difference between the effectiveness of the two techniques. Many of the modules studied contained branches with no statements so there *were* differences in the test case sets (noticeable by the difference in the mean number of test cases required for each technique), although only one module/fault (ID=7) provided a different probability of detection and even in this case the fault did not manifest itself in the 'statementless' branch (the difference in effectiveness was simply due to the extra test cases required to exercise the branches with no statements).

The closeness of the results for statement and branch testing was not surprising as the only additional faults directly targetted by branch (over statement) testing are in the execution of the 'statementless' branches, which had already been exercised in the original testing.

6.4 Condition testing techniques

Five of the (fourteen) modules with faults contained multiple conditions, but in no cases was the multiple condition a cause of failure. Where no multiple conditions appeared in a module, then the probabilities of detection remained constant for branch, BC, MCDC and BCC testing. For modules with multiple conditions it was simply the number of extra test cases required to satisfy the different condition testing techniques that caused the slightly higher test effectiveness results. Statistically, neither measure provided any justification for rejecting the null hypotheses (that there was no significant difference in the effectiveness of compared techniques).

One module/fault (ID=12) contained a multiple condition but maintained the same value for probability of detection for all white box techniques. This was explained by the distribution of statements on branches earlier in the module, the execution of which required all combinations of the later multiple condition to be executed - so no extra test cases were required.

The author found that the extra complexity of deriving a minimal test set to satisfy MCDC, compared to deriving the simpler, albeit larger, test set required for BCC, appeared to make the use of MCDC rather than BCC pointless - especially as the probability of detection for MCDC never surpassed that for BCC for any module.

6.5 Random testing

According to the mean values for probability of detection of faults then BVA was clearly far more effective than random testing, and the statistical measures to confirm this. Due to the diminishing returns provided by increasing the number of random test cases then without automated test oracles random testing cannot compete with BVA. Over 50,000 random tests per module were required to achieve the same mean probability as BVA.

Random testing fared well in comparison to all the other techniques studied, and in no case did the statistical measures provide sufficient evidence to discount the null hypothesis that the compared techniques provided similar levels of test effectiveness. With just six test cases per module, random testing would have achieved a higher level of effectiveness than EP and any of the white box techniques considered. Given that random testing requires much less effort than these other techniques to generate the test input values, then random testing compared very favourably.

6.6 Limitations of the experiment

The use of the concepts underlying domain testing and symbolic execution to define the fault-finding and test case sets means that the study has been restricted in a similar manner to these techniques. For instance, only modules with linear predicates could be analysed, however all the modules studied satisfied these constraints, a situation reflected in symbolic execution studies [21]. One way of surmounting possible problems in future studies may be to use automatic random test input generation to produce large numbers of inputs from within each test case set and to subsequently measure the proportion that cause the module to fail. The same technique could be used to validate the analytically-determined probabilities calculated from the 'overlap' of the polyhedra used in this study.

It is recognised that the identification of equivalence partitions, which is required for both EP and BVA, can be somewhat subjective. This could have caused some bias, but the high level of detail provided by the module specifications and the use of well-defined procedures should have mitigated this risk.

The assumption that each of the test techniques would be applied to achieve 100% coverage will not hold for all projects, however the author believes that the setting of reduced levels of coverage should be discouraged. For instance, the setting of criteria of less than 100% coverage for EP and BVA is impractical. All equivalence partitions and boundaries must be identified to calculate the coverage achieved by testing a subset and there would be little justification for then not testing to 100%. The setting of required white box test coverage levels of less than 100% will often lead to the more complex parts of a module not being exercised and the author believes that these parts will generally harbour a disproportionately large number of faults. Traditional arguments for allowing reduced levels of coverage due to the inability of achieving a strictly-defined complete coverage need not apply if the definitions in [6] are used, which allow infeasible code to be discounted from the calculations (as long as it is both identified and documented).

7. Conclusions

This paper has presented the results of a study into the test effectiveness of a number of testing techniques. The experimental methodology used here has differed from previous experiments as the derived test effectiveness is based on an analysis of *all* inputs that satisfy the test technique rather than arbitrarily chosen inputs from this set. The results, in summary, are:

- BVA was the most effective technique studied, achieving a highest mean probability of detection of 0.75, compared with 0.16 for EP and 0.17 for the most effective white box technique studied (BCC). However, to achieve this, nearly twice as many test cases were required (on average, 13.7 for BVA, 5.1 for EP and 4.7 for BCC).
- For lower levels of effectiveness, random testing appears to have the advantage over both EP and all the white box techniques considered. Just six random test cases per module are required to achieve a better level of effectiveness than any of these techniques, and random testing will also be less expensive to perform as no test case design is required.
- Where multiple conditions are present and condition testing is worthwhile, the use of MCDC rather than BCC appears pointless due to the extra effort required to define the test case sets, and the (albeit marginally) reduced levels of test effectiveness.
- Although random testing appears efficient for achieving lower levels of test effectiveness, it requires a prohibitive number of test cases (over 50,000) to reach the levels achievable by BVA.
- If choosing a black box technique to complement branch testing then BVA appears to be more effective than either EP, or random testing.
- It would appear that reliance on a single test technique is not the best approach as even BVA would always have missed faults, and so the identification of which techniques complement each other most effectively appears to warrant further study.

Obviously the results of this study have taken little account of cost, except in general terms; measures could not be taken as the techniques were not used to detect faults, but rather to create test case sets. More study of the costs of applying testing techniques is required, although the relatively large differences in test effectiveness results presented here may well overshadow relatively small differences in the cost of their application.

Using the same example avionics system, the methodology has still be applied to other testing techniques, such as those based on data flow. The next step will then be to re-apply the methodology to other systems to determine if the results are system-specific.

As with all such experiments the results are characterised by the conditions under which it took place (described in Appendix A) and their extrapolation to other environments is not implied.

8 References

[1] Roper, M. et al, ‘Towards the Experimental Evaluation of Software Testing Techniques, *Proc. EuroSTAR ‘94*, Brussels, Oct. 1994.

[2] Frankl, P.G. and Weiss, S.N. 'An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing' *IEEE Trans. on Software Eng.*, Vol. 19, No. 8, pp. 774-787, 1993.

[3] Chan, F. T., Chen, T.Y., Mak, I.K. and Yu, Y.T. ‘Practical considerations in using partition testing’, *Proc. SQM ‘94*, Edinburgh, July 1994.

[4] Weyuker, E.J. and Jeng, B., ‘Analyzing Partition Testing Strategies’, *IEEE Trans. on Software Engineering*, Vol. 17, No. 7, July 1991, pp 703-711.

[5] Hamlet, R. and Taylor, R., ‘Partition Testing Does Not Inspire Confidence’, *IEEE Trans. on Software Eng.*, Vol. 16, No. 12, December 1990, pp 1402-1411.

[6] 'BCS Software Component Testing Standard', Issue 3.3, April 1997 (available from the author).

[7] Basili, V.R. and Selby, R.W. 'Comparing the Effectiveness of Software Testing Strategies', *IEEE Trans. on Software Eng.*, Vol.SE-13, No.12, pp. 1278-1296, Dec. 1987.

[8] Myers, G.J. 'A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections', *Comms. of the ACM*, Vol. 21, No. 9, pp. 760-768, 1978.

[9] Jones, C., 'Applied Software Measurement: Assuring Productivity and Quality', McGraw-Hill, NY, 1991.

[10] Jeng, B., and Weyuker, E.J., 'Some Observations on Partition Testing', *Proc. ACM SIGSOFT 3rd Symposium on Software Testing, Analysis, and Verification,* ACM Press, Dec 1989, pp 38-47.

[11] Chen, T.Y. and Yu, Y.T., 'On the Expected Number of Failures Detected by Subdomain Testing and Random Testing', *IEEE Trans. on Software Eng.*, Vol. 22, No. 2, pp. 109-119, Feb. 1996.

[12] Hamlet, R. 'Theoretical Comparison of Testing Methods', *SIGSOFT Software Eng. Notes*, Vol. 14, Iss.8, pp. 28-37, 1989.

[13] Ntafos, S.C. 'A Comparison of Some Structural Testing Strategies', *IEEE Trans. on Software Eng.*, Vol. 14, No.6, pp. 868-874, June 1988.

[14] Duran, J.W. and Ntafos, S.C., 'An Evaluation of Random Testing', *IEEE Trans. on Software Eng.*, Vol. SE-10, No. 4, July 1984, pp 438-444.

[15] Weyuker, E.J., 'More Experience with Data Flow Testing', *IEEE Trans. on Software Eng.*, Vol. 19, No. 9, Sep. 1993, pp 912-919.

[16] Selby, R.W., 'Combining Software Testing Strategies: An Empirical Evaluation', *Proc. of the Workshop on Software Testing*, Banff, Canada, 15-17 July 1986.

[17] Girgis, M.R. and Woodward, M.R., 'An Experimental Comparison of the Error Exposing Ability of Program Testing Criteria', *Proc. of the Workshop on Software Testing,* Banff, Canada, 15-17 July 1986.

[18] Hetzel, W.C., 'Making Software Measurement Work', *QED Publishing Group*, 1993.

[19] Reid, S.C., 'Test Effectiveness in Software Module Testing', *Proc. EuroSTAR '94*, Brussels, Oct. 1994.

[20] Cooper, B.E., 'Statistics for Experimentalists', *Pergamon Press*, 1969.

[21] Coward, P. D., 'Symbolic Execution Systems - A Review', *Software Engineering Journal*, Vol. 3, Part 6, 1988, pp 229-239.

[22] Weyuker, E.J. and Frankl, P.G., 'A Formal Analysis of the Fault-Detecting Ability of Testing Methods', *IEEE Trans. on Software Eng.*, Vol. 19, No. 3, March 1993, pp 202-213.

[23] Weyuker, E.J. and Frankl, P.G., 'Provable Improvements on Branch Testing', *IEEE Trans. on Software Engineering*, Vol. 19, No. 10, October 1990, pp 962-975.

[24] Vouk, M.A., Tai, K.C. and Paradkar, A., 'Empirical Studies of Predicate-Based Software Testing', *Proc. Fifth International Symposium on Software Reliability Engineering*, Monterey, Nov. 1994.

Appendix A - Characterising the study

The chosen case study project was a military aircraft display system using Motorola 68020 and Intel 80186 processors, developed to DO178A level 2 (essential). The final system comprised approximately 20,000 lines of Ada code (executable statements) and 3,500 lines of assembler. Six software engineers were involved in the project, with between 2 and 14 years experience, at an average of 6½ years and development took nearly two years.

The notation used for the module specifications was structured english. The level of abstraction of the specifications was generally low, just above the source code.

The implementation language was Ada 83. The project coding standard mandated that neither GOTO statements nor tasking should be used and subprograms should not exceed 200 lines of code. The average number for lines of code per module studied was 78.5.

The project required that functional testing was carried out and a number of test coverage levels be achieved, although from discussion with the testers, and from project documentation, it became apparent that functional testing was not performed. Statement and branch coverage levels of 100% were required, while LCSAJ coverage to 60% was required, and achieved by default.