Boundary Value Analysis

Blake Neate
327966

Contents

1.0 Introduction

The practice of testing software has become one of the most important aspects of the process of software creation. When we are testing software the first and potentially most crucial step is to design test cases. There are many methods associated with test case design. This report will document the approach known as Boundary Value analysis (BVA).

As the incredibly influential Dijkstra stated “Testing can show the presence of bugs, but not the absence”. Although this is true we find that testing can be very good at the first, if implemented correctly. For this reason we need to know of the techniques available so we can find the correct method for the system under test (SUT).

We will look at the various topics associated with Boundary Value Analysis and use some simple examples to show their meaning and purpose. There will be some examples to show the usefulness of each method. There will be an ongoing “small scale” example to help picture each method. This will be accompanied by two examples introduced by P.C. Jorgensen [1]. These will be used to show some more “true to life” requirements for testing techniques. There will be a chapter detailing test cases for these two more in-depth examples.

2.0 The Testing Problem

Developing effective and efficient testing techniques has been a major problem when creating test cases; this has been the point of discussion for many years. There are several well known techniques associated with creating test cases for a system.

There are many issues that can undermine the integrity of the result from and given test suite (set of tests) implementation. These issues or questions can be as basic as where do we start? They can become more complicated when we try to ascertain where testing should end and if we have covered all the required permutations.

3.0 The Typing Of Languages

The typing of languages can have a large bearing on the effect of the Boundary Value Analysis approach. Strongly typed languages such as PASCAL and ADA require that all constants or variables defined must have an associated data type, which dictates the data ranges of these values upon definition.

A large reason for languages like these to be created was to prevent the nature of errors that Boundary Value Analysis is used to discover. Although BVA is not completely ineffective when used in conjunction with languages of this nature, BVA can be seen as unsuitable for systems created using them.

Boundary Value Analysis is therefore more suitable to more "free-form" languages such as COBOL and FORTRAN which are not so strongly typed. These are also known as weak typing languages and can be seen as languages which allow one type (i.e. a String) to be seen as another (i.e. an Int). This can be useful but it can also cause bugs. These bugs or errors are normally found in the ranges that BVA operates in and therefore can find.

4.0 The Focus of BVA

Boundary Value Analysis focuses on the input variables of the function. For the purposes of this report I will define two variables (I will only define two so that further examples can be kept concise) X_1 and X_2. Where X_1 lies between A and B and X_2 lies between C and D.

$A \leq X_1 \leq B$
$C \leq X_2 \leq D$

The values of A, B, C and D are the extremities of the input domain. These are best demonstrated by figure 4.1.

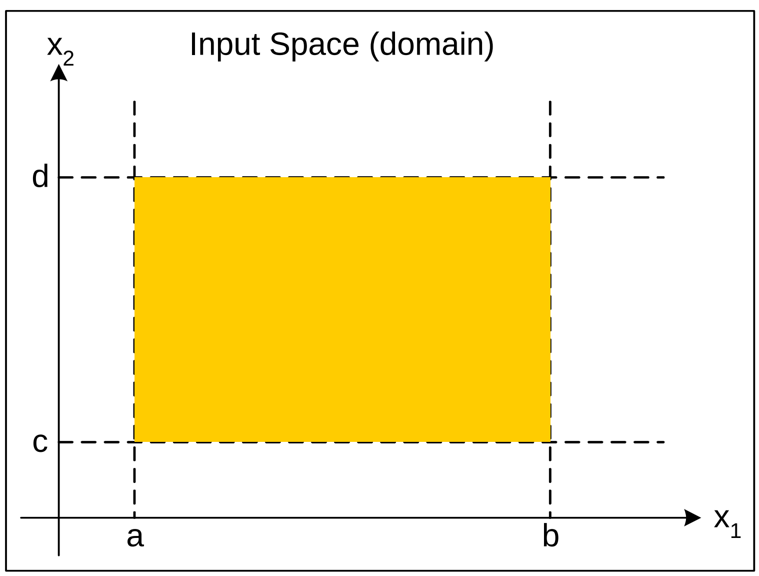

Figure 4.1

The Yellow shaded area of the graph sho … ble/legitimate input domain of the given function. As the name suggests Boundary Value Analysis focuses on the boundary of the input space to recognize test cases. The idea and motivation behind BVA is that errors tend to occur near the extremities of the input variables. The defects found on the boundaries of these input variables can obviously be the result of countless possibilities.

But there are many common faults that result in errors more collated towards the boundaries of input variables. For example if the programmer forgot to count from zero or they just miscalculated. Errors in the code concerning loop counters being off by one or the use of a < operator instead of ≤. These are all very common mistakes and accompanied with other common errors we find an increasing need to perform Boundary Value Analysis.

5.0 Applying Boundary Value Analysis

In the general application of Boundary Value Analysis can be done in a uniform manner. The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are:

- Min ------------------------------------- Minimal
- Min+ ------------------------------------- Just above Minimal
- Nom ------------------------------------- Average
- Max- ------------------------------------- Just below Maximum
- Max ------------------------------------- Maximum

In continuing our example this results in the following test cases shown in figures 5.1 and 5.2:

$<x_{1nom}, x_{2min}>$ $<x_{1nom}, x_{2min+}>$ $<x_{1nom}, x_{2nom}>$ $<x_{1nom}, x_{2max-}>$ $<x_{1nom}, x_{2max}>$
$<x_{1min}, x_{2nom}>$ $<x_{1min+}, x_{2nom}>$ $<x_{1max}, x_{2nom}>$ $<x_{1max}, x_{2nom}>$

Figure 5.1

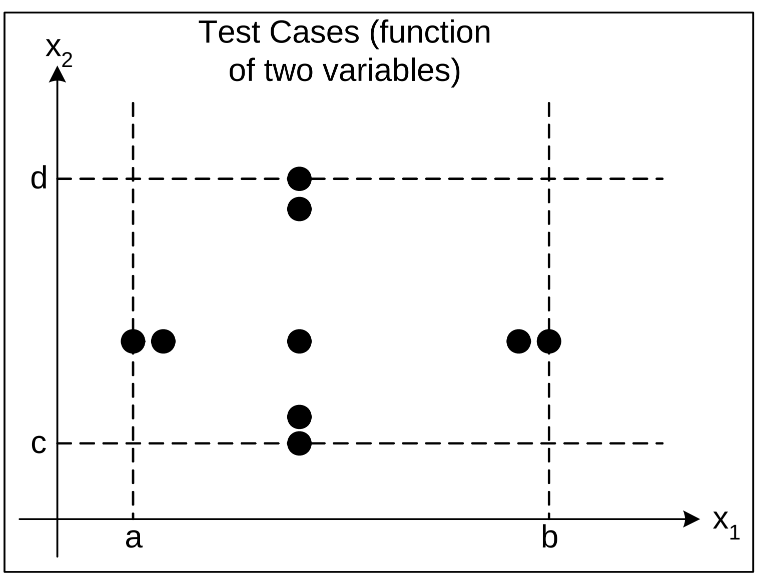

Figure 5.2

You maybe wondering why it is we are only concerned with one of the values taking on their extreme values at any one particular time. The reason for this is that generally Boundary Value Analysis uses the **Critical Fault Assumption**. There are advantages and shortcomings of this method. The advantages will be discussed in chapter 5.2, and alternative methods will be shown in chapter 7.

5.1 Some Important examples

To be able to demonstrate or explain the need for certain methods and their relative merits I will introduce two testing examples proposed by P.C. Jorgensen [1]. These examples will provide more extensive ranges to show where certain testing techniques are required and provide a better overview of the methods usability.

- The NextDate problem

The NextDate problem is a function of three variables: day, month and year. Upon the input of a certain date it returns the date of the day after that of the input.
The input variables have the obvious conditions:

$1 \leq \text{Day} \leq 31$.
$1 \leq \text{month} \leq 12$.
$1812 \leq \text{Year} \leq 2012$.
(Here the year has been restricted so that test cases are not too large).

There are more complicated issues to consider due to the dependencies between variables. For example there is never a 31st of April no matter what year we are in. The nature of these dependencies is the reason this example is so useful to us. All errors in the NextDate problem are denoted by "Invalid Input Date."

- The Triangle problem

In fact the first introduction of the Triangle problem is in 1973, Gruenburger. There have been many more references to this problem since making this one of the most popular example to be used in conjunction with testing literature.

The triangle problem accepts three integers (a, b and c)as its input, each of which are taken to be sides of a triangle. The values of these inputs are used to determine the type of the triangle (Equilateral, Isosceles, Scalene or not a triangle).

For the inputs to be declared as being a triangle they must satisfy the six conditions:

C1. $1 \leq a \leq 200$.
C2. $1 \leq b \leq 200$.
C3. $1 \leq c \leq 200$.

C4. $a < b + c$.
C5. $b < a + c$.
C6. $c < a + b$.

Otherwise this is declared not to be a triangle.
The type of the triangle, provided the conditions are met, is determined as follows:

1. If all three sides are equal, the output is Equilateral.
2. If exactly one pair of sides is equal, the output is Isosceles.
3. If no pair of sides is equal, the output is Scalene.

5.2 Critical Fault Assumption

The Critical Fault Assumption also known as the single fault assumption in reliability theory. The assumption relies on the statistic that failures are only rarely the product of two or more simultaneous faults. Upon using this assumption we can reduce the required calculations dramatically.

The amount of test cases for our example as you can recall was 9. Upon inspection we find that the function ***f*** that computes the number of test cases for a given number of variables ***n*** can be shown as:

$$f = 4n + 1$$

As there are four extreme values this accounts for the 4***n.*** The addition of the constant one constitutes for the instance where all variables assume their nominal value.

5.3 Generalising BVA

There are two approaches to generalising Boundary Value Analysis. We can do this by the number of variables or by the ranges these variables use. To generalise by the number of variables is relatively simple. This is the approach taken as shown by the general Boundary Value Analysis technique using the critical fault assumption.

Generalizing by ranges depends on the type of the variables. For example in the NextDate example proposed by P.C. Jorgensen [1], we have variable for the year, month and day. Languages similar to the likes of FORTRAN would normally encode the month's variable so that January corresponded to 1 and February corresponded to 2 etc. Also it would be possible in some languages to declare an enumerated type {Jan, Feb, Mar,……, Dec}. Either way this type of declaration is relatively simple because the ranges have set values.

When we do not have explicit bounds on these variable ranges then we have to create our own. These are know as artificial bounds and can be illustrated via the use of the Triangle problem. The point raised by P.C. Jorgensen was that we can easily impose a lower bound on the length of an edge for the tri-angle as an edge with a negative length would be “silly”. The problem occurs when trying to decide upon an upper bound for the length of each length. We could use a certain set integer, we could allow the program to use the highest possible integer (normally denoted as something to the effect of MaxInt). The arbitrary nature of this problem can lead to messy results or non concise test cases.

5.4 Limitations of BVA

Boundary Value Analysis works well when the Program Under Test (PUT) is a “function of several independent variables that represent bounded physical quantities” [1]. When these conditions are met BVA works well but when they are not we can find deficiencies in the results.

For example the NextDate problem, where Boundary Value Analysis would place an even testing regime equally over the range, tester’s intuition and common sense shows that we require more emphasis towards the end of February or on leap years.

The reason for this poor performance is that BVA cannot compensate or take into consideration the nature of a function or the dependencies between its variables. This lack of intuition or understanding for the variable nature means that BVA can be seen as quite rudimentary.

6.0 Robustness Testing

Robustness testing can be seen as and extension of Boundary Value Analysis. The idea behind Robustness testing is to test for clean and dirty test cases. By clean I mean input variables that lie in the legitimate input range. By dirty I mean using input variables that fall just outside this input domain.

In addition to the aforementioned 5 testing values (min, min+, nom, max-, max) we use two more values for each variable (min-, max+), which are designed to fall just outside of the input range.

If we adapt our function $\boldsymbol{f}$ to apply to Robustness testing we find the following equation:

$$\boldsymbol{f} = 6\boldsymbol{n} + 1$$

I have equated this solution by the same reasoning that lead to the standard BVA equation. Each variable now has to assume 6 different values each whilst the other values are assuming their nominal value (hence the $6\boldsymbol{n}$), and there is again one instance whereby all variables assume their nominal value (hence the addition of the constant 1). These result can be seen in figures 6.1 and 6.2.

Robustness testing ensues a sway in interest, where the previous interest lied in the input to the program, the main focus of attention associated with Robustness testing comes in the expected outputs when and input variable has exceeded the given input domain. For example the NextDate problem when we an entry like the 31st June we would expect an error message to the effect of "that date does not exist; please try again".

Robustness testing has the desirable property that it forces attention on exception handling. Although Robustness testing can be somewhat awkward in strongly typed languages it can show up altercations. In Pascal if a value is defined to reside in a certain range then and values that falls outside that range result in the run time errors that would terminate any normal execution. For this reason exception handling mandates Robustness testing.

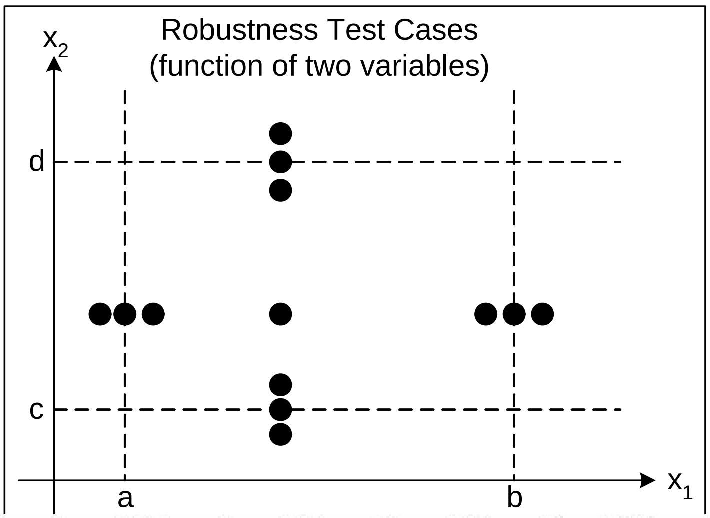

Figure 6.1

$<x_{1nom}, x_{2min}><x_{1nom}, x_{2min+}><x_{1nom}, x_{2nom}><x_{1nom}, x_{2max-}><x_{1nom}, x_{2max}><x_{1min}, x_{2nom}><x_{1min+}, x_{2nom}>$
$<x_{1nom}, x_{2nom}><x_{1max}, x_{2nom}><x_{1max}, x_{2nom}><x_{1min}, x_{2nom}><x_{1nom}, x_{2max+}><x_{1max+}, x_{2nom}>$

Figure 6.2

7.0 Worst-Case Testing

Boundary Value analysis uses the critical fault assumption and therefore only tests for a single variable at a time assuming its extreme values. By disregarding this assumption we are able to test the outcome if more than one variable were to assume its extreme value. In an electronic circuit this is called Worst Case Analysis. In Worst-Case testing we use this idea to create test cases.

To generate test cases we take the original 5-tuple set (min, min+, nom, max-, max) and perform the Cartesian product of these values. The end product is a much larger set of results than we have seen before.

We can see from the results in figures 7.1 and 7.2 that worst case testing is a more comprehensive testing technique. This can be shown by the fact that standard Boundary Value Analysis test cases are a proper subset of Worst-Case test cases.

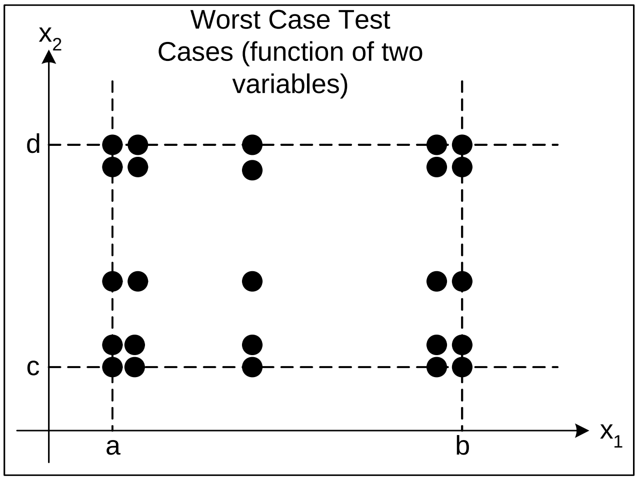

Figure 7.1

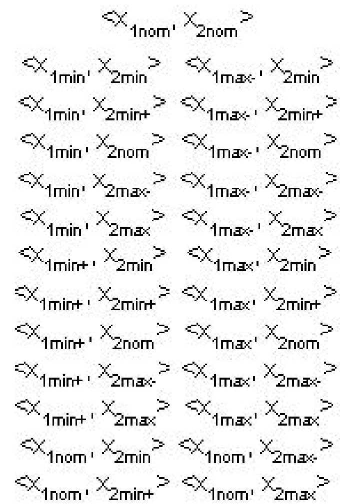

Figure 7.2

These test cases although more comprehensive in their coverage, constitute much more endeavour. To compare we can see that Boundary Value Analysis results in 4***n*** + 1 test case where Worst-Case testing results in 5^n test cases. As each variable has to assume each of its variables for each permutation (the Cartesian product) we have 5 to the ***n*** test cases.

For this reason Worst-Case testing is generally used for situations that require a higher degree of testing (where failure of the program would be very costly)with less regard for the time and effort required as for many situations this can be too expensive to justify.

7.1 Robust Worst-Case Testing

If the function under test were to be of the greatest importance we could use a method named Robust Worst-Case testing which as the name suggests draws it attributes from Robust and Worst-Case testing.

Test cases are constructed by taking the Cartesian product of the 7-tuple set defined in the Robustness testing chapter. Obviously this results in the largest set of test results we have seen so far and requires the most effort to produce.

We can see that the function ***f*** (to calculate the number of test cases required) can be adapted to calculate the amount of Robust Worst-Case test cases. As there are now 7 values each variable can assume we find the function ***f*** to be:

$$f = 7^n$$

This function has also been reached in the paper A Testing and analysis tool for Certain 3-Variable functions [2].

The results for the continuing example can be seen in figures 7.3 and 7.4.

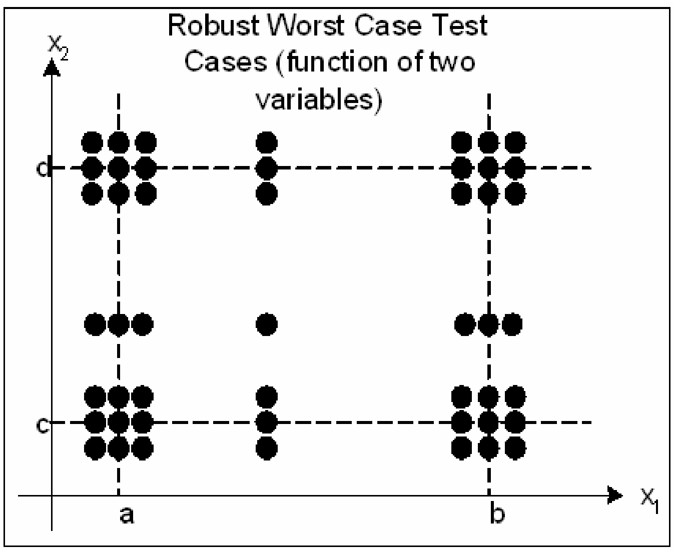

Figure 7.3

$<X_{1min-}, X_{2min-}>$	$<X_{1min}, X_{2max}>$	$<X_{1nom}, X_{2nom}>$	$<X_{1max-}, X_{2min}>$
$<X_{1min-}, X_{2min}>$	$<X_{1min}, X_{2max+}>$	$<X_{1nom}, X_{2max-}>$	$<X_{1max-}, X_{2min+}>$
$<X_{1min-}, X_{2min+}>$	$<X_{1min+}, X_{2min-}>$	$<X_{1nom}, X_{2max}>$	$<X_{1max-}, X_{2nom}>$
$<X_{1min-}, X_{2nom}>$	$<X_{1min+}, X_{2min}>$	$<X_{1nom}, X_{2max+}>$	$<X_{1max-}, X_{2max-}>$
$<X_{1min-}, X_{2max-}>$	$<X_{1min+}, X_{2min+}>$	$<X_{1max+}, X_{2min-}>$	$<X_{1max-}, X_{2max}>$
$<X_{1min-}, X_{2max}>$	$<X_{1min+}, X_{2nom}>$	$<X_{1max+}, X_{2min}>$	$<X_{1max-}, X_{2max+}>$
$<X_{1min-}, X_{2max+}>$	$<X_{1min+}, X_{2max-}>$	$<X_{1max+}, X_{2min+}>$	$<X_{1max}, X_{2min-}>$
$<X_{1min}, X_{2min-}>$	$<X_{1min+}, X_{2max}>$	$<X_{1max+}, X_{2nom}>$	$<X_{1max}, X_{2min}>$
$<X_{1min}, X_{2min}>$	$<X_{1min+}, X_{2max+}>$	$<X_{1max+}, X_{2max-}>$	$<X_{1max}, X_{2min+}>$
$<X_{1min}, X_{2min+}>$	$<X_{1nom}, X_{2min-}>$	$<X_{1max+}, X_{2max}>$	$<X_{1max}, X_{2nom}>$
$<X_{1min}, X_{2nom}>$	$<X_{1nom}, X_{2min}>$	$<X_{1max+}, X_{2max+}>$	$<X_{1max}, X_{2max-}>$
$<X_{1min}, X_{2max-}>$	$<X_{1nom}, X_{2min+}>$	$<X_{1max-}, X_{2min-}>$	$<X_{1max}, X_{2max}>$
			$<X_{1max}, X_{2max+}>$

Figure 7.4

8.0 Examples: Test Cases

For each example I will show test cases for the standard Boundary Value Analysis and the Worst-case testing techniques. These will show how the test cases are performed and how comprehensive the results are. There will not be test cases for Robustness testing or robust Worst-case testing as the cases covered should explain how the process works. Too many test cases would prove to be monotonous when trying to explain a concept, however when presenting a real project when the figures are more "necessary" all test cases should be detailed and explained to their full extent.

8.1 Next Date problem

Standard Boundary Value Analysis test cases:

month	day	year
min = 1	min = 1	min = 1812
min+ = 2	min+ = 2	min+ = 1813
nom = 6	nom = 15	nom = 1912
max- = 11	max- = 30	max- = 2011
max = 12	max = 31	max = 2012

Boundary Value Analysis Test Cases

Case	month	day	year	Expected Output
1	6	15	1812	June 16, 1812
2	6	15	1813	June 16, 1813
3	6	15	1912	June 16, 1912
4	6	15	2011	June 16, 2011
5	6	15	2012	June 16, 2012
6	6	1	1912	June 2, 1912
7	6	2	1912	June 3, 1912
8	6	30	1912	July 1, 1912
9	6	31	1912	error
10	1	15	1912	January 16, 1912
11	2	15	1912	February 16, 1912
12	11	15	1912	November 16, 1912
13	12	15	1912	December 16, 1912

Worst-Case Analysis test cases:

Worst Case Test Cases (60 of 125)				
Case	month	day	year	Expected Output
1	1	1	1812	January 2, 1812
2	1	1	1813	January 2, 1813
3	1	1	1912	January 2, 1912
4	1	1	2011	January 2, 2011
5	1	1	2012	January 2, 2012
6	1	2	1812	January 3, 1812
7	1	2	1813	January 3, 1813
8	1	2	1912	January 3, 1912
9	1	2	2011	January 3, 2011
10	1	2	2012	January 3, 2012
11	1	15	1812	January 16, 1812
12	1	15	1813	January 16, 1813
13	1	15	1912	January 16, 1912
14	1	15	2011	January 16, 2011
15	1	15	2012	January 16, 2012
16	1	30	1812	January 31, 1812
17	1	30	1813	January 31, 1813
18	1	30	1912	January 31, 1912
19	1	30	2011	January 31, 2011
20	1	30	2012	January 31, 2012
21	1	31	1812	February 1, 1812
22	1	31	1813	February 1, 1813
23	1	31	1912	February 1, 1912
24	1	31	2011	February 1, 2011
25	1	31	2012	February 1, 2012
26	2	1	1812	February 2, 1812
27	2	1	1813	February 2, 1813
28	2	1	1912	February 2, 1912
29	2	1	2011	February 2, 2011
30	2	1	2012	February 2, 2012
31	2	2	1812	February 3, 1812
32	2	2	1813	February 3, 1813
33	2	2	1912	February 3, 1912
34	2	2	2011	February 3, 2011
35	2	2	2012	February 3, 2012
36	2	15	1812	February 16, 1812
37	2	15	1813	February 16, 1813
38	2	15	1912	February 16, 1912
39	2	15	2011	February 16, 2011
40	2	15	2012	February 16, 2012
41	2	30	1812	error
42	2	30	1813	error
43	2	30	1912	error
44	2	30	2011	error
45	2	30	2012	error
46	2	31	1812	error
47	2	31	1813	error
48	2	31	1912	error
49	2	31	2011	error
50	2	31	2012	error
51	6	1	1812	June 2, 1812
52	6	1	1813	June 2, 1813
53	6	1	1912	June 2, 1912
54	6	1	2011	June 2, 2011
55	6	1	2012	June 2, 2012
56	6	2	1812	June 3, 1812
57	6	2	1813	June 3, 1813
58	6	2	1912	June 3, 1912
59	6	2	2011	June 3, 2011
60	6	2	2012	June 3, 2012

As we can see there are only 60 of 125 test cases in this example, this shows the vast amount of test cases produced.

8.2 Tri-angle problem

Standard Boundary Value Analysis test cases:

min = 1
min+ = 2
nom = 100
max- = 199
max = 200

Boundary Value Analysis Test Cases				
Case	a	b	c	Expected Output
1	100	100	1	Isosceles
2	100	100	2	Isosceles
3	100	100	100	Equilateral
4	100	100	199	Isosceles
5	100	100	200	Not a Triangle
6	100	1	100	Isosceles
7	100	2	100	Isosceles
8	100	199	100	Isosceles
9	100	200	100	Not a Triangle
10	1	100	100	Isosceles
11	2	100	100	Isosceles
12	199	100	100	Isosceles
13	200	100	100	Not a Triangle

Worst-Case Analysis test cases:

Worst Case Test Cases (60 of 125)

Case	a	b	c	Expected Output	Case	a	b	c	Expected Output
1	1	1	1	Equilateral	31	2	2	1	Isosceles
2	1	1	2	Not a Triangle	32	2	2	2	Equilateral
3	1	1	100	Not a Triangle	33	2	2	100	Not a Triangle
4	1	1	199	Not a Triangle	34	2	2	199	Not a Triangle
5	1	1	200	Not a Triangle	35	2	2	200	Not a Triangle
6	1	2	1	Not a Triangle	36	2	100	1	Not a Triangle
7	1	2	2	Isosceles	37	2	100	2	Not a Triangle
8	1	2	100	Not a Triangle	38	2	100	100	Isosceles
9	1	2	199	Not a Triangle	39	2	100	199	Not a Triangle
10	1	2	200	Not a Triangle	40	2	100	200	Not a Triangle
11	1	100	1	Not a Triangle	41	2	199	1	Not a Triangle
12	1	100	2	Not a Triangle	42	2	199	2	Not a Triangle
13	1	100	100	Isosceles	43	2	199	100	Not a Triangle
14	1	100	199	Not a Triangle	44	2	199	199	Isosceles
15	1	100	200	Not a Triangle	45	2	199	200	Scalene
16	1	199	1	Not a Triangle	46	2	200	1	Not a Triangle
17	1	199	2	Not a Triangle	47	2	200	2	Not a Triangle
18	1	199	100	Not a Triangle	48	2	200	100	Not a Triangle
19	1	199	199	Isosceles	49	2	200	199	Scalene
20	1	199	200	Not a Triangle	50	2	200	200	Isosceles
21	1	200	1	Not a Triangle	51	100	1	1	Not a Triangle
22	1	200	2	Not a Triangle	52	100	1	2	Not a Triangle
23	1	200	100	Not a Triangle	53	100	1	100	Isosceles
24	1	200	199	Not a Triangle	54	100	1	199	Not a Triangle
25	1	200	200	Isosceles	55	100	1	200	Not a Triangle
26	2	1	1	Not a Triangle	56	100	2	1	Not a Triangle
27	2	1	2	Isosceles	57	100	2	2	Not a Triangle
28	2	1	100	Not a Triangle	58	100	2	100	Isosceles
29	2	1	199	Not a Triangle	59	100	2	199	Not a Triangle
30	2	1	200	Not a Triangle	60	100	2	200	Not a Triangle

Again this is only up to 60 of 125 test cases.

9.0 Conclusion

As Glenford J. Myers [3] summarises, we can find that Boundary Value Analysis “if practised correctly, is one of the most useful test-case-design methods”. But he goes on to say that it is often used ineffectively as the testers often see it as so simple they misuse it, or don’t use it to its full potential. This is a very true interpretation of the use of Boundary Value Analysis.

BVA can provide a relatively simple and formal testing technique that can be very powerful when used correctly. When issues arise such as dependencies between variables or a need for foresight into the system’s functionality, we can find Boundary Value Analysis restrictive (as shown by the NextDate problem).

The underlying fact is that generally Boundary Value Testing techniques are computationally and theoretically inexpensive in the creation of test cases. For this reason in many cases it can be desirable in its results to effort ratio. This means that Boundary Value Analysis still has a part to play in modern day testing practises and should be wit us for some time to come.

10.0 References

[1] P. Jorgenson, Software Testing- A Craftsman's Approach, CRC Press, New York, 1995

[2] Naryan C Debnath, Mark Burgin, Haesun K. Lee, Eric Thiemann, A Testing and analysis tool for Certain 3-Variable functions, Winona State University.

[3] Glenford J. Myers, The Art of Software Testing, John Wiley and Sons, Inc. 2004