Boundary Value Analysis According to the ISTQB® Foundation Level Syllabus

Matthias Hamburg, German Testing Board, matthias.hamburg@istqb.org

Adam Roman, Polish Testing Board, Jagiellonian University, adam.roman@uj.edu.pl

Boundary value analysis (BVA) is one of the basic and most widely used test techniques by testers. The reason for the popularity of this technique is that it is relatively simple, and at the same time very effective in finding a common type of defects. However, there are misunderstandings about its application. The purpose of this article is to explain how exactly to apply this technique according to ISTQB®.

1. The technique

The ISTQB® Foundation Syllabus (v4.0) describes the BVA as follows:

> *Boundary Value Analysis (BVA) is a technique based on exercising the boundaries of equivalence partitions. Therefore, BVA can only be used for ordered partitions. The minimum and maximum values of a partition are its boundary values. In the case of BVA, if two elements belong to the same partition, all elements between them must also belong to that partition.*
>
> *BVA focuses on the boundary values of the partitions because developers are more likely to make errors with these boundary values. Typical defects found by BVA are located where implemented boundaries are misplaced to positions above or below their intended positions or are omitted altogether.*
>
> *This syllabus covers two versions of the BVA: 2-value and 3-value BVA. They differ in terms of coverage items per boundary that need to be exercised to achieve 100% coverage.*
>
> *In 2-value BVA (Craig 2002, Myers 2011), for each boundary value there are two coverage items: this boundary value and its closest neighbor belonging to the adjacent partition. To achieve 100% coverage with 2-value BVA, test cases must exercise all coverage items, i.e., all identified boundary values. Coverage is measured as the number of boundary values that were exercised, divided by the total number of identified boundary values, and is expressed as a percentage.*
>
> *In 3-value BVA (Koomen 2006, O'Regan 2019), for each boundary value there are three coverage items: this boundary value and both its neighbors. Therefore, in 3-value BVA some of the coverage items may not be boundary values. To achieve 100% coverage with 3-value BVA, test cases must exercise all coverage items, i.e., identified boundary values and their neighbors. Coverage is measured as the number of boundary values and their neighbors exercised, divided by the total number of identified boundary values and their neighbors, and is expressed as a percentage.*
>
> *3-value BVA is more rigorous than 2-value BVA as it may detect defects overlooked by 2-value BVA. For example, if the decision "if (x ≤ 10) …" is incorrectly implemented as "if (x = 10) …", no test data derived from the 2-value BVA (x = 10, x = 11) can detect the defect. However, x = 9, derived from the 3-value BVA, is likely to detect it.*

2. Boundary values are not the same as borders

Boundary values are not the same as the border between two equivalence partitions. In common understanding, a border lies *between* two areas separating them, just like a geographical border lies between two geographical territories. In contrast, boundary values are elements of a specific equivalence partition marking its limits. In mathematics, the notion of a boundary can be defined precisely, but it needs a good understanding of the basic concepts of general topology. However, for the purpose of software testing, ISTQB® Foundation Level uses the simpler case of ordered partitions in which the minimum and maximum values constitute the boundaries.

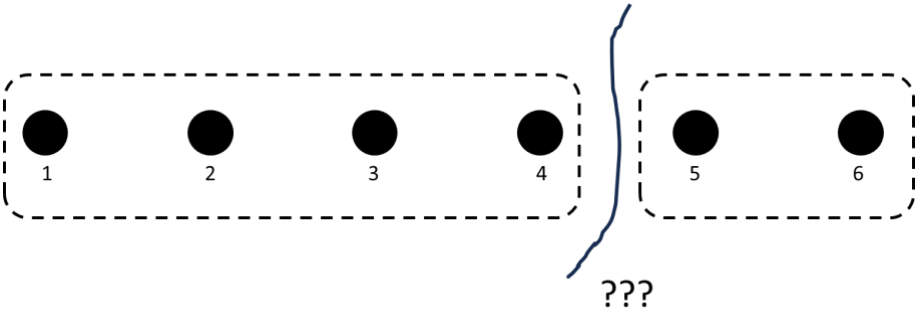

Figure 1: Border and boundaries for a simple example

Suppose, for example, that the test object has the six-element domain $\{1, 2, 3, 4, 5, 6\}$ from the figure above as input and that it shall behave differently on the two partitions marked with rounded rectangles: $\{1, 2, 3, 4\}$ and $\{5, 6\}$. In software testing, there is no point in talking about the "border *between* the two equivalence partitions." If we want to verify that the boundaries are implemented correctly, it rather makes sense to test the behavior at (or around) the boundary values of each equivalence partition: values 1 and 4 for the first one, and values 5 and 6 for the second one.

3. BVA step by step

The application of the BVA follows the procedure below.

Step 1. Identify the value domain
Step 2. Determine the equivalence partitions
Step 3. Identify the boundary values of the equivalence partitions
Step 4. For each boundary value, determine the coverage items according to the variants of the BVA (2-value or 3-value coverage)
Step 5. Determine the union of all coverage item sets, taking into account that they may overlap
Step 6. Design test cases for the identified coverage items

Example. *The registration form of a certain program contains a field "login". A valid login has at least 6 and at most 15 characters.*

We want to apply BVA to verify whether this function is implemented correctly.

Step 1. The domain for which we want to verify the implementation is the *length of the login*. Thus, it will be an ordered, infinite set of positive integers: $D = \{0, 1, 2, 3, 4, \ldots\}$.

Step 2. Valid logins have lengths from 6 to 15, so we distinguish the equivalence partition "lengths of correct login": $P = \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. The remaining elements form two separate partitions: $Q = \{0, 1, 2, 3, 4, 5\}$ (login too short) and $R = \{16, 17, 18, \ldots\}$ (login too long). In total, we have three equivalence partitions.

Step 3. The boundary values of the equivalence partitions are as follows:

- The boundary values of *Q* are 0 and 5.
- The boundary values of *P* are 6 and 15.
- The boundary value of *R* is 16 (there is no maximum value in *R*, so *R* has only one boundary value).

Step 4. In 2-value BVA, for each boundary value there are two coverage items: the boundary value itself and its neighbor belonging to an adjacent partition (see the figure below, in which black points represent the coverage items in 2-value BVA). Note that the set of all coverage items is the same as the set of all boundary values.

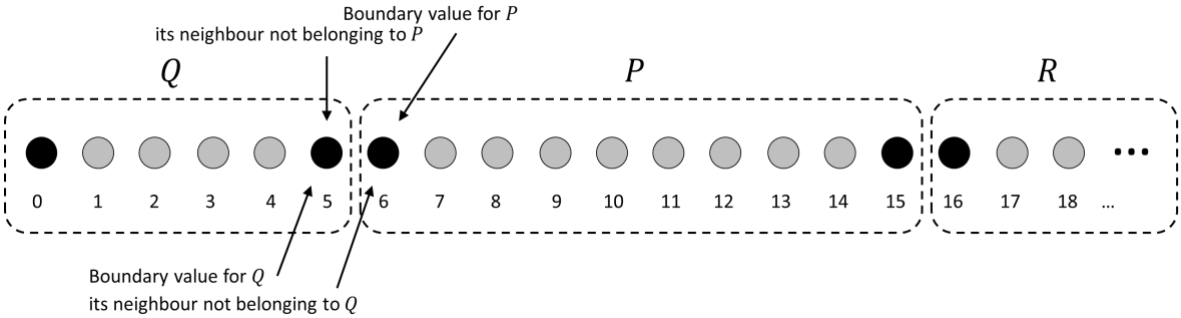

Figure 2: 2-value BVA coverage items for the login example

In 3-value BVA, for each boundary value there are three coverage items: this boundary value and both its neighbors, regardless to which partition they belong (see the figure below, in which white points represent the additional coverage items in 3-value BVA).

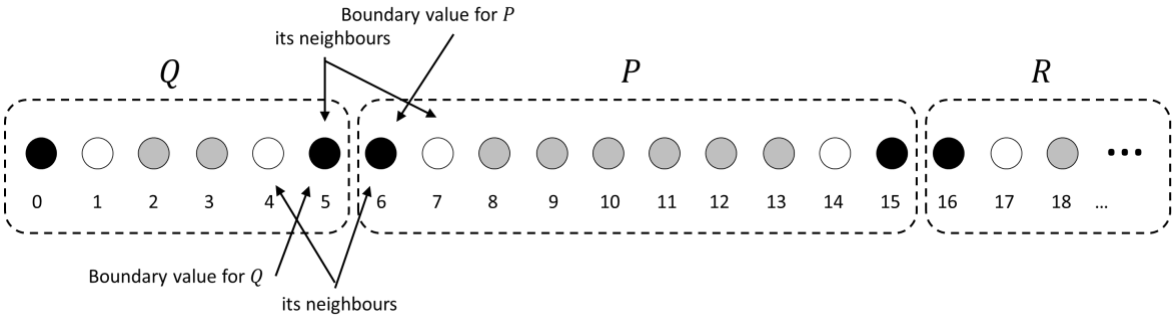

Figure 3: 3-value BVA coverage items for the login example

The table below summarizes the coverage items:

Partition	Boundary value	2-value BVA coverage items	3-value BVA coverage items
Q	0	0	0, 1
	5	5, 6	4, 5, 6
P	6	6, 5	5, 6, 7
	15	15, 16	14, 15, 16
R	16	16, 15	15, 16, 17

Table 1: Coverage items for the login example

Note that there is no coverage item –1 for the boundary value 0, because this is an infeasible string length.

Step 5. The union of the coverage items is:

- In 2-value BVA: {0, 5, 6, 15, 16}
- In 3-value BVA: {0, 1, 4, 5, 6, 7, 14, 15, 16, 17}

These are the coverage items that need to be tested.

Step 6. For each coverage item we design a separate test case, shown in the tables below. Note that the input is the string that represents a login. Each such string corresponds to an element of the considered domain *D*, representing the length of this login.

TC	Test input (login)	Expected result	Coverage item
TC-Q-min	*<empty string>*	Login rejected (too short)	0
TC-Q-max	Abcde	Login rejected (too short)	5
TC-P-min	John99	Login accepted	6
TC-P-max	123456789012345	Login accepted	15
TC-R-min	SixteenCharLogin	Login rejected (too long)	16

Table 2: Test cases for the login example in 2-value BVA

TC	Test input (login)	Expected result	Coverage item
TC-Q-min+1	x	Login rejected (too short)	1
TC-Q-max-1	p0qr	Login rejected (too short)	4
TC-P-min+1	XsevenX	Login accepted	7
TC-P-max-1	12345678901234	Login accepted	14
TC-R-min+1	VeryVeryLongLogin	Login rejected (too long)	17

Table 3: Additional test cases for the login example in 3-value BVA

4. The number of coverage items

It is a common misconception that 2-value BVA requires twice as many coverage items as borders, and 3-value BVA requires three times as many. Both are wrong, as illustrated in the example above.

In 2-value BVA, the standard case is that there are two coverage items at a border separating two adjacent partitions (one on each side). See for example the border between *Q* and *P* in the example above. However, in special cases there are less coverage items:

- at an outer border, there is only one coverage item (e.g. 0 in the example above);
- if an equivalence partition contains only one value, the two adjacent borders will require in total three coverage items rather than four because of the overlap.

In 3-value BVA, the standard case is that there are *four* coverage items at a border separating two adjacent partitions. See, again, the border between *Q* and *P* in the example above. However, there are again special cases like for 2-value BVA with less than four coverage items for a border. This is the case at outer borders and for borders at both sides of a partition with less than four elements.

5. Defect types addressed by the BVA

BVA is a technique that focuses on defects in domain implementation. To illustrate its effectiveness in domain defect detection, let us consider the login example above.

A developer is tasked with implementing the login validation. To verify if the login is long enough, the developer might design the following code snippet:

```
IF length ≤ 5 THEN
      Login is too short
ELSE
      Login is not too short
END IF
```

Alternatively, the developer might also write:

```
IF length < 6 THEN
      Login is too short
ELSE
      Login is not too short
END IF
```

Both implement the specification correctly. However, the developer could also pick a wrong operator from the set {<, ≤, =, ≠, ≥, >} of possible operators. This would result in a common type of defect in the implementation of domains. Let us see which BVA coverage items would detect which of these defects.

In 2-value BVA the coverage items are 5 and 6 as indicated above. 3-value BVA adds the coverage items 4 and 7. So we can list the actual results of the two IF predicates for each operator mentioned above and compare them with the correct result:

IF predicate	Actual results for the coverage items:			
	5	6	4	7
Length ≤ 5 (correct)	True	False	True	False
Length < 5 (incorrect)	False	False	True	False
Length = 5 (incorrect)	True	False	False	False
Length ≠ 5 (incorrect)	False	True	True	True
Length ≥ 5 (incorrect)	True	True	False	True
Length > 5 (incorrect)	False	True	False	True
Length < 6 (correct)	True	False	True	False
Length ≤ 6 (incorrect)	True	True	True	False
Length = 6 (incorrect)	False	True	False	False
Length ≠ 6 (incorrect)	True	False	True	True
Length ≥ 6 (incorrect)	False	True	False	True
Length > 6 (incorrect)	False	False	False	True

Table 4: Actual results for the login minimal length in case of correct and incorrect predicate implementations (green = as expected, red = not as expected)

As we can see, the inputs 5 and 6 for 2-value BVA detect 8 of the 10 incorrect predicates. However, they fail to detect two cases of defective predicates: Length = 5 and Length ≠ 6. These defects would pass the 2-value BVA tests. Depending on the risk level, the effort for the two additional coverage items for the 3-value BVA (4 and 7) might pay off. This example also shows that 3-value BVA is stronger than 2-value BVA but requires more coverage items.

6. Dealing with seamless transitions

BVA is a black-box technique. During test execution, the tester must be able to recognize from the actual results whether the test object behaves according to the correct or incorrect equivalence

partition. This may not be the case with seamless transitions of the functionality. In such cases, it is a good idea to define an additional equivalence partition in which the expected behaviors are indistinguishable. This allows for the verification of the implementation of the boundary values with black-box testing. The following example illustrates this situation.

A banking application should charge a fee for trades depending on the trade amount in Euros. The fee is 1% of the trade amount, but no less than 1€. The fee is rounded to the nearest Euro cent.

In this example, the input value domain consists of non-negative numbers with the precision of two decimal places. The specification indicates two equivalence partitions:

- EP1 for amounts between 0.00€ and 100.00€ for which the fee is 1€;
- EP2 for amounts greater than 100.00€ for which the fee is 1% of the amount.

However, neither a 2-value BVA black-box test with the inputs 100.00€ and 100.01€, nor a 3-value one with the additional inputs 99.99€ and 100.02€ will indicate if the partitions have been correctly implemented. For all these inputs, the fee will be 1.00€ both for the 1% rate and for the constant. Hence, we adjust the two original partitions and introduce an additional equivalence partition for the transition area:

- EP1' adjusted to amounts between 0.00€ and 99.49€ for which the fee is 1€ and not 1% of the amount;
- EP2' for amounts greater than or equal 100.50€ for which the fee is 1% of the amount and not 1€;
- EP3' for amounts from 99.50€ to 100.49€ for which the fee is 1€.

For the new EP1', the boundary value of 99.49€ is the maximum for which the 1% rate would yield an incorrect fee of 0.99€. Similarly, for the new EP2', the amount of 100.50 € is the minimum value for which the 1€ fee would be incorrect. These boundary values will have the greatest chance to detect implementation defects. For example, these tests will detect the faulty implementation "IF amount < 101 THEN fee = 1€ ELSE fee = 1% * amount", which is undetected with the boundary values for the old partitions EP1, EP2.

7. Further practical considerations

We conclude this paper with some further practical considerations.

Clear value domains. It is not without reason that we mention the identification of the value domain as the first step of the BVA. Each type of variable has its own value domain. For example, if the input is the length of a string, it will be a non-negative integer. It will usually be beyond the scope of testing to check if the test object handles negative length in a specific way. In general, the tester should differentiate between possible inputs (within the value domain) and impossible ones (for which the behavior of the test object does not need to be specified). Within the domain of all possible inputs, the tester may differentiate between valid inputs (for which the test object delivers a business value) and invalid inputs (which should be rejected by the test object in a specified manner).

Precise constraints. When analyzing a specification for boundary values, the tester should pay great attention to the way domain constraints are specified. This is because constraints can be formulated in natural language, using phrases such as: "at least", "at most", "less than", "more than", "from x to y", "between x and y", etc., which might be misinterpreted. A review of the test basis using an appropriate checklist for boundary value analysis is recommended.

Continuous domains. In the real world, we can deal with continuous value domains (for example, the speed of a rocket is a real number in a continuous interval). In practice, the representation of numbers in a computer is finite, so it is necessary to choose an appropriate level of precision and perform analysis taking into account this specified level of precision.

ON, OFF, IN and OUT values. Rather than considering boundary values and their neighbors on both sides, some authors consider values ON the boundary (the boundary value itself), OFF the boundary (the closest neighbor belonging to the adjacent partition), IN values (inside the equivalence partition but not being the ON point), and OUT values (outside the equivalence partition but not being the OFF point). The only difference between BVA and this technique is that IN and OUT values are not necessarily neighboring the boundary values. For one-dimensional value domains, using ON and OFF values is equivalent to 2-value BVA.

Multiple dimensions. In practice, the input domain of the test object is often multidimensional, determined by more than one variable, and its behavior depends on a combination of multiple inputs. For equivalence partitions of multidimensional domains, the boundary value analysis from the ISTQB® Foundation Level Syllabus is not readily applicable. Such situations require an extension of the technique based on ON, OFF, IN and OUT values, which is called *domain analysis*. Domain analysis goes beyond the scope of this paper.