Black-box test design: Input space partitioning

Black-box testing

- Test design against a specification, no implementation is needed
- Test values are derived from the specification or type information
- For each test value, the expected result is derived from the specification
  - This must always be the case: If you use source code to derive the expected result it means you understand the code, not that the code works!

Faults may still remain in unexecuted component code, so black box testing needs to be complemented by white box testing after the implementation is available

Input Domains

The input domain for a program contains all the possible inputs to that program

- For even small programs, the input domain can be large or infinite

Testing is fundamentally about choosing finite sets of values from the input domain

Input parameters define the scope of the input domain

- Parameters to a method
- Data read from a file
- Global variables
- User level inputs

Domain for each input parameter is partitioned into regions of essentially equivalent values

At least one value is chosen from each region

Benefits of Input Space Partitioning

- Can be equally applied at several levels of testing
  - Unit
  - Integration
  - System

- Relatively easy to apply with no automation

- Easy to adjust the procedure to get more or fewer tests

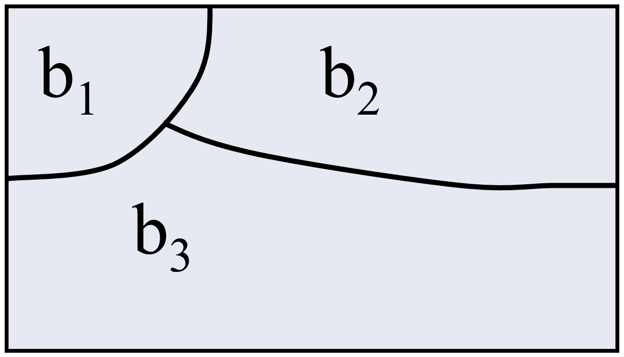
- No implementation knowledge is needed
  - just the interface for the input space

Partitioning Domains

- Domain D
- A partition *of D* defines a *set of (finite)* *blocks*, *$B = \{ b_1, b_2, \dots b_n \}$* that satisfy two properties :

  1. blocks must be *pairwise disjoint* (no overlap)
  2. together the blocks *cover* the complete domain *D*

  Blocks are equivalence classes!



1. $b_i \cap b_j = \emptyset, \quad \forall\, i \neq j,\ b_i, b_j \in B$

2. $b_1 \cup \dots \cup b_Q = D$

Using Partitions – Assumptions

- Choose a value from each block
- Each value is assumed to be equally useful for testing

- Application to testing
  - Find characteristics in the inputs : parameters, semantic descriptions, …
  - Partition each characteristic
  - Choose tests by combining values from characteristics

- Example characteristics
  - Array (null, zero-length, arbitrary length)
  - Order of the input file (sorted, inverse sorted, arbitrary, …)
  - Input device (DVD, CD, VCR, computer, …)

Choosing Partitions

Choosing (or defining) partitions seems easy, but is easy to get it wrong

Consider the "*order of array a*"

b_1 = sorted in ascending order

b_2 = sorted in descending order

b_3 = arbitrary order

but … something is fishy …

What if the file is of length 1?

The file will be in all three blocks …

That is, disjointness is not satisfied

Solution

Each characteristic should address just one property

Array a sorted ascending
- b1 = true
- b2 = false

Array a sorted descending
- b1 = true
- b2 = false

Properties of Partitions

- If the partitions are not complete or disjoint, that means the partitions have not been considered carefully enough
- They should be reviewed carefully, like any design attempt
- Different alternatives should be considered
- We model the input domain in five steps …

Modeling the Input Domain

Step 1 : Identify testable functions

- Individual methods have one testable function
- In a class, each method often has the same characteristics
- Programs have more complicated characteristics
- Systems of integrated hardware and software components can use devices, operating systems, hardware platforms, browsers, etc

Modeling the Input Domain

Step 2 : Find all the parameters

- Often fairly straightforward, even mechanical
- Important to be complete
- Methods : Parameters and state (non-local) variables used
- Components : Parameters to methods and state variables
- System : All inputs, including files and databases

Modeling the Input Domain

Step 3 : Model the input domain

- The domain is scoped by the parameters
- The structure is defined in terms of characteristics
- Each characteristic is partitioned into sets of blocks
- Each block represents a set of values

This is the most creative design step in using ISP

Candidate characteristics:

- Relationship of variables with special values (zero, null, blank, …)
- Relationships among variables

Modeling the Input Domain (*cont*)

Step 4 : Apply a test criterion to choose combinations of values

- A test input has a value for each parameter
- One block for each characteristic
- Choosing all combinations is usually infeasible
- Coverage criteria allow subsets to be chosen

Step 5 : Refine combinations of blocks into test inputs

- Choose appropriate values from each block

Two Approaches

Input Domain Model (IDM)

1. Interface-based approach
   - Develops characteristics directly from individual input parameters
   - Simplest application
   - Can be partially automated in some situations

2. Functionality-based approach
   - Develops characteristics from a behavioral view of the program under test
   - Harder to develop—requires more design effort
   - May result in better tests, or fewer tests that are as effective

Interface-Based Approach

- Mechanically consider each parameter in isolation
- This is an easy modeling technique and relies mostly on syntax
- No use of domain and semantic information
  - Could lead to an incomplete IDM
- Ignores relationships among parameters

Consider the method private static int triang (int Side1, int Side2, int Side3)
// A triangle classification method returning number of side of a triangle that are equals, error otherwise

IDM for each parameter is identical

Reasonable characteristics : *Relation of side with zero, return value is 1,2,3, others*

Interface-Based Approach: characteristics

- Range: one characteristic with values inside the range, and two with values outside the range
  - For example, let speed∈[60 .. 90].
  - Then, we generate three blocks <60, [60..90], > 90
  - input tests: 50, 75, and 92.

- String: at least one containing all legal strings and one containing all illegal strings.
  - For example, let fname: string be a variable to denote a first name.
  - Then, we could generate the following blocks: Empty, Letters only and <20 , alphanumeric and <20, strings longer than 20 charachters
  - Input tests: λ, Carl-Ludwig Heinz, X-AE-A12, ThisIsAVeryLongNameTooLongToBeTrue

Interface-Based Approach: characteristics

- Enumeration: Each value in a separate block
  - For example, consider auto_color ∈ {red, blue, green}.
  - The following blocks are considered red, blue, green with the obvious test inputs

- Array: One block containing all legal arrays, one containing only the empty array, and one containing arrays larger than the expected size
  - For example, consider int[] aName = new int [3].
  - The following blocks can be automatically generated: Empty, array of 1 tot 3 integers, array with more than 3 integers
  - Examples of input tests are [], [-10, 20], [-9, 0, 12, 15].

Functionality-Based Approach

- Identify characteristics that correspond to the intended functionality
  - Can incorporate domain and semantic knowledge
  - Can use relationships among parameters
- Modeling can be based on requirements, but not on implementation
- The same parameter may appear in multiple characteristics, so it's harder to translate values to test cases

Consider the method private static int triang (int Side1, int Side2, int Side3)
// A triangle classification method returning number of side of a triangle that are equals, error otherwise

IDM can combine all parameters

Reasonable characteristic : *Type of triangle, return value is 1, 2, 3, others*

Characteristics and blocks: example

```
public boolean findElement (List list, Object element)
            // Effects: if list or element is null throw NullPointerException
            //              else return true if element is in the list, false otherwise
```

Interface-Based Approach
Two parameters : list, element
Characteristics :
  list, element is null (block1 = true, block2 = false)
  list is empty (block1 = true, block2 = false)
  return value (block1 = true, block2 = false)

Characteristics and blocks: example

```
public boolean findElement (List list, Object element)
        // Effects: if list or element is null throw NullPointerException
        //          else return true if element is in the list, false otherwise
```

Functionality-Based Approach

Two <u>parameters</u> : list, element

<u>Characteristics</u> :

- number of occurrences of element in list (block1 = 0, block2 = 1, block3 = >1)
- element occurs first in list (block1 = true, block2 = false)
- element occurs last in list (block1 = true, block2 = false)
- return value (block1 = true, block2 = false)

Triang example: Interface-Based IDM

private static int Triang (int Side1, int Side2, int Side3)
// A triangle classification method, returning number of side of a triangle that are equals, error otherwise

Three int parameters

Characteristic	b_1	b_2	b_3	b_4
"Relation of Side 1 to 0"	>= 0	< 0		
"Relation of Side 2 to 0"	>= 0	< 0		
"Relation of Side 3 to 0"	>= 0	< 0		
Return value	1	2	3	≠ 1, 2, 3

Input errors

A maximum of 2*2*2 = 8 tests, but some triangles are valid, some are invalid

Refining the characterization can lead to more tests …

Triang example: Interface-Based IDM

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

- A maximum of 3*3*3 = 27 tests
- This is only complete because the inputs are integers

Possible values for partition q_1			
Characteristic	b_1	b_2	b_3
Side1	5	0	-5

Triang example: Functionality-Based IDM

Characteristic	b_1	b_2	b_3	b_4
q_1 = "type of triangle"	scalene	isosceles	equilateral	invalid

Oops … not a partition… equilateral is also isosceles !

We need to refine the example to make characteristics valid

Characteristic	b_1	b_2	b_3	b_4
q_1 = "type of triangle "	scalene	Isosceles not equilateral	equilateral	invalid

Possible values for partition q_1				
Characteristic	b_1	b_2	b_3	b_4
Type of triangle	(4, 5, 6)	(3, 3, 4)	(2, 2, 2)	(3, 4,15)

Choosing Combinations of Values

- Once characteristics and partitions are defined, the next step is to choose test values
- We use coverage criteria – to choose effective subsets
- The most obvious criterion is to choose all combinations …

All Combinations (AC) : All combinations of blocks from all characteristics must be used.

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

Too many: the second characterization of Triang results in 3*3*3 = 27 tests

Example AC

All Combinations (AC) : All combinations of blocks from all characteristics must be used.

- Three partitions with blocks [A, B], [1, 2, 3], and [x, y]

  AC consists of the following twelve (=2x3x2) tests:

  - (A, 1, x) (B, 1, x) (A, 2, x) (B, 2, x) (A, 3, x) (B, 3, x)
  - (A, 1, y) (B, 1, y) (A, 2, y) (B, 2, y) (A, 3, y) (B, 3, y)

ISP Criteria – Each Choice

27 tests for Triang is maybe too much. The other extreme is to test only one value from each block

Each Choice (EC) : One value from each block for each characteristic must be used in at least one test case.

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

ISP Criteria – Each Choice

27 tests for Triang is maybe too much. The other extreme is to test only one value from each block

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

For Triang this means 3 tests only: (4, 5, 2), (0, 0,0), and (-5, -3, -6)

- For three partitions with blocks [A, B], [1, 2, 3], and [x, y]
- EC will include, for example: (A, 1, x), (B, 2, y), (A, 3, x)

ISP Criteria – Pair-Wise

- Each choice yields few tests – cheap but perhaps ineffective

- Another approach asks values to be combined with other values

Pair-Wise (PW) : A value from each block for each characteristic must be combined with a value from every block for each other characteristic.

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

Example PW

Pair-Wise (PW) : A value from each block for each characteristic must be combined with a value from every block for each other characteristic.

- Three partitions with blocks [A, B], [1, 2, 3], and [x, y] PW needs to cover the pairs:

(A, 1)	(A, 2)	(A, 3)	(A, x)	(A, y)	
(B, 1)	(B, 2)	(B, 3)	(B, x)	(B, y)	
(1, x)	(1, y)	(2, x)	(2, y)	(3, x)	(3, y)

- And this can be done, for example, by the triples

(A, 1, x)	(A, 2, x)	(A, 3, x)
(B, 1, y)	(B, 2, y)	(B, 3, y)
(A, –, y)	(B, –, x)	

ISP Criteria – Pair-Wise

- Each choice yields few tests – cheap but perhaps ineffective
- Another approach asks values to be combined with other values

Pair-Wise (PW) : A value from each block for each characteristic must be combined with a value from every block for each other characteristic.

For Triang this means 9 tests

For Triang:	2, 2, 2	2, 0, 0	2, -3, -3
	0, 2, 0	0, 0, 2	0, -2, 2
	-5, 4, -3	-5, 0, 1	-5, -5, 0

ISP Criteria –n-Wise

A natural extension is to require combinations of *n* values instead of *2*

n-Wise (nW) : A value from each block for each group of n characteristics must be combined.

If *n* is the number of characteristics, then *n-Wise = AC*

That is … *n*-Wise can be expensive and benefits are not clear

ISP Criteria – Base Choice

Testers sometimes recognize that certain values are important using domain knowledge

Base Choice (BC) : A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic.

Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	2 greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	2 greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	2 greater than 1	between -1 and 1	less than -1

ISP Criteria – Base Choice

- Testers sometimes recognize that certain values are important
- This uses domain knowledge of the program

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1 2	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1 2	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1 2	between -1 and 1	less than -1

For Triang: Base	2, 2, 2	2, 2, 0	2, 0, 2	0, 2, 2
		2, 2, -3	2, -5, 2	-4, 2, 2

Example PW

Pair-Wise (PW) : A value from each block for each characteristic must be combined with a value from every block for each other characteristic.

Three partitions with blocks [A, B], [1, 2, 3], and [x, y].

Assume base choice blocks are ‘A’, ‘1’ and ‘x’.

- The base choice test is (A, 1, x)
- The additional tests could be

(B, 1, x) (A, 2, x) (A, 3, x) (A, 1, y)

Base Choice Notes

- The base test must be feasible
  - That is, all base choices must be compatible
- Base choices can be
  - Most likely from an end-use point of view
  - Simplest
  - Smallest
  - First in some ordering
- The base choice is a crucial design decision
  - Test designers should document why the choices were made

ISP Criteria – Multiple Base Choice

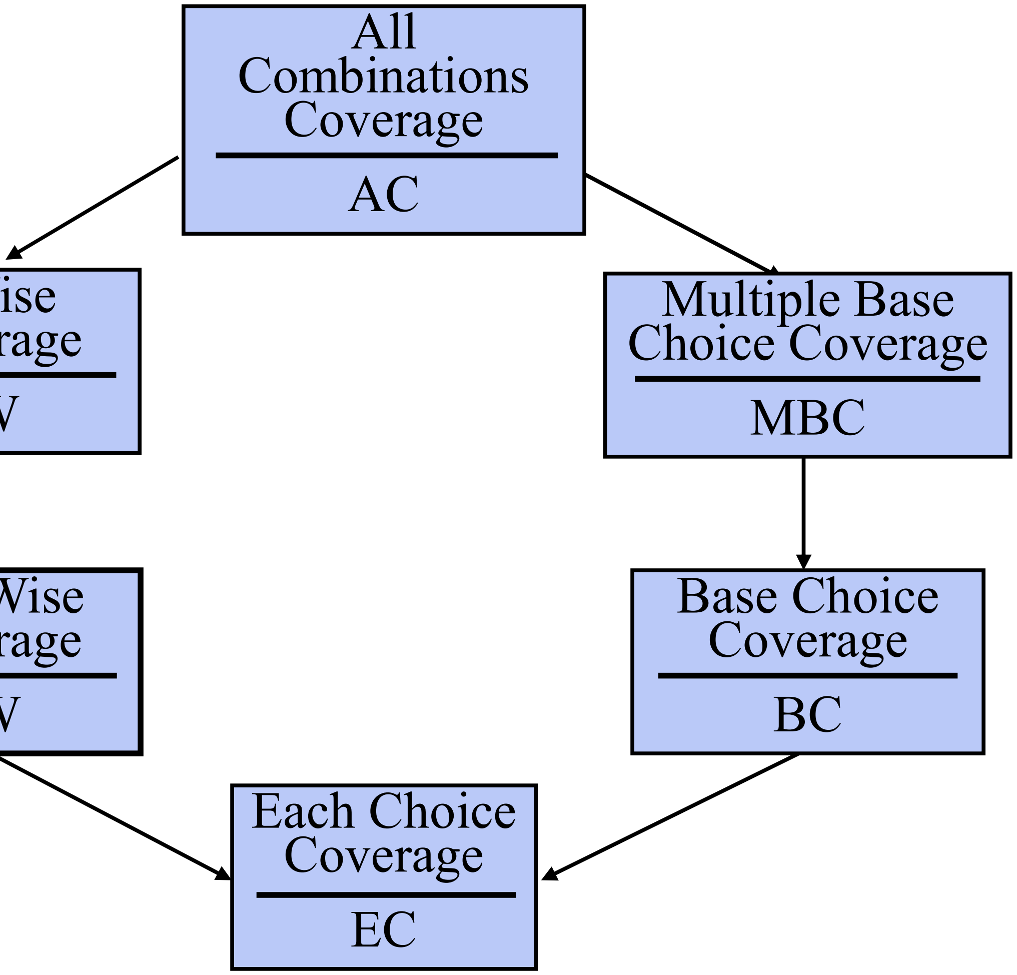
Multiple Base Choice (MBC) : One or more base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic.

Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choices in each other characteristic.

For Triang:

base 1	2, 2, 2	2, 2, 0	2, 0, 2	0, 2, 2
		2, 2, -3	2, -5, 2	-4, 2, 2
base 2	1, 1, 1	1, 1, 0	1, 0, 1	0, 1, 1
		1, 1, -4	1, -3, 1	-2, 1, 1

ISP Coverage Criteria Subsumption



Constraints Among Characteristics

- Some combinations of blocks are infeasible
  - "less than zero" and "scalene" … not possible at the same time
- These are represented as constraints among blocks
- Two general types of constraints
  - A block from one characteristic cannot be combined with a specific block from another
  - A block from one characteristic can ONLY BE combined with a specific block form another characteristic
- Handling constraints depends on the criterion used
  - AC, PW, nW : Drop the infeasible pairs
  - BC, MBC : Change a value to another non-base choice to find a feasible combination

Input Space Partitioning - Summary

- Analysis:
  - find all implicit and explicit parameters
  - Find value range for each parameter

- Design:
  - Select suitable equivalence partitions
  - Choose coverage criteria
  - Select test values for coverage criteria and calculate expected result
    - Choose values (possibly randomly) in the middle of partition, not at boundary
    - Eliminate redundant test cases
  - If needed add test cases to cover all partitions of the return value/output parameters

Input Space Partitioning - Evaluations

- Strengths
  - Fairly easy to apply, good basic level of black-box testing
  - Applicable to all levels of testing – unit, class, integration, system, etc.
  - Structured mean based only on the input space of the program, not on its implementation

- Weaknesses
  - Correctness at the boundary is not tested
  - Interface based approach does not test combination of inputs

Simple, straightforward, effective, and widely used in practice

Black-box test design:
Boundary value analysis

Boundary values analysis

- A test selection technique that targets faults in applications at the boundaries of equivalence classes.
  1. Partition the input domain
  2. Identify the boundaries values for each partition
     - Each partition has two boundary values
  3. Select test data such that each boundary value occurs in at least one test input

Characteristic	b_1		b_2		b_3	
"Relation of Side 1 to 0"	greater than 1		between -1 and 1		less than -1	
Boundary values	int.MAX_VALUE	2	-1	1	-2	int.MIN_VALUE

Boundary values

- A boundary value is a value at the boundary of an equivalence partition.
- Each partition has two boundary values

Characteristic	b_1		b_2		b_3	
“Relation of Side 1 to 0”	greater than 1		between -1 and 1		less than -1	
Boundary values	int.MAX_VALUE	2	-1	1	-2	int.MIN_VALUE

Coverage criteria

- Use *All Combination* (AC) to cover all normal boundary values
- Use *Each Choice* (EC) to cover the error test cases

Characteristic	b_1		b_2		b_3	
Boundary values Side 1	int.MAX_VALUE	2	-1	1	-2	int.MIN_VALUE
Boundary values Side 2	int.MAX_VALUE	2	-1	1	-2	int.MIN_VALUE
Boundary values Side 3	int.MAX_VALUE	2	-1	1	-2	int.MIN_VALUE

For Triang this means 4*4*4 = 64 normal tests values + 2 error tests values

If 64 tests are too much…you can substitute AC with PW or BC, for example

Boundary value analysis - Summary

- Analysis:
  - Identify boundary values for each partition (no overlap, no gap)

- Design:
  - Identify normal and error boundary values
  - Choose coverage criteria for normal boundary value
  - Select normal test values for coverage criteria and error test values for EC coverage
  - Eliminate redundant test cases
  - Calculate expected result

Boundary value analysis - Evaluations

- Strengths
  - Test values are immediately given by the boundary values
  - Focus on area where faults are more likely

- Weaknesses
  - It doubles the number of test cases for each partition
  - Combinations of values are not tested

Simple, straightforward, effective, and widely used in combination with input space partition

Black-box test design:
Decision table testing

Combining test values

- Decision tables are a systematic way to combine test values from different inputs supplementing the interface based approach when we use weaker coverage than *All choices* (AC)

- No domain knowledge needed as in the functional based approach

- A decision table is a model of functional requirements mapping combinations of input values (causes) with matching output values (effect) through rules

Causes

- Causes are normal partitions seen as predicates. They can be either true or false

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 1	between -1 and 1	less than -1
q_2 = "Relation of Side 2 to 0"	greater than 1	between -1 and 1	less than -1
q_3 = "Relation of Side 3 to 0"	greater than 1	between -1 and 1	less than -1

Causes	Rules							
Side1 = 1	T	T	T	T	F	F	F	F
Side1 > 1	F	F	F	F	T	T	T	T
Side2 = 1	T	T	F	F	T	T	F	F
Side2 > 1	F	F	T	T	F	F	T	T
Side3 = 1	T	F	T	F	T	F	T	F
Side3 > 1	F	T	F	T	F	T	F	T

Error partitions and unfeasible combination are removed

- Both Side1 = 1 and Side1 > 1 are T

Effect

- Effects are normal partitions of outpus seen as predicates.

Effect	Rules							
Return value =1	~~T~~	~~T~~	~~T~~	T	F	F	F	F
Return value = 2	~~T~~	~~T~~	~~F~~	F	T	T	F	F
Return value = 3	~~T~~	~~F~~	~~T~~	F	T	F	T	F

Error partitions and unfeasible combination are removed

- Both Return value = 1, 2 and 3

Combining causes and effect

- Causes and effect are combined verifying the feasibility of the combination against the specification. For example Side1=1, Side2=1 and Side3=1 (EQI) all true can only result in return value=3)

Causes	Rules							
Side1 = 1	T	T	T	T	F	F	F	F
Side1 > 1	F	F	F	F	T	T	T	T
Side2 = 1	T	T	F	F	T	T	F	F
Side2 > 1	F	F	T	T	F	F	T	T
Side3 = 1	T	F	T	F	T	F	T	F
Side3 > 1	F	T	F	T	F	T	F	T
Effect								
Return value =1	F							
Return value = 2	F							
Return value = 3	T							

Combining causes and effect

- Two sides =1 and the other side not can result in return value = 2 (ISO) or all false

Causes	Rules										
Side1 = 1	T	T	T	T	T	T	F	F	F	F	F
Side1 > 1	F	F	F	F	F	F	T	T	T	T	T
Side2 = 1	T	T	T	F	F	F	T	T	T	F	F
Side2 > 1	F	F	F	T	T	T	F	F	F	T	T
Side3 = 1	T	F	F	T	T	F	T	T	F	T	F
Side3 > 1	F	T	T	F	F	T	F	F	T	F	T
Effect											
Return value =1	F	F	F	F	F		F	F			
Return value = 2	F	T	F	T	F		T	F			
Return value = 3	T	F	F	F	F		F	F			

Combining causes and effect

- Two sides =1 and the other side not can result in return value = 2 (ISO) or all false

Causes	Rules										
Side1 = 1	T	T	T	T	T	T	F	F	F	F	F
Side1 > 1	F	F	F	F	F	F	T	T	T	T	T
Side2 = 1	T	T	T	F	F	F	T	T	T	F	F
Side2 > 1	F	F	F	T	T	T	F	F	F	T	T
Side3 = 1	T	F	F	T	T	F	T	T	F	T	F
Side3 > 1	F	T	T	F	F	T	F	F	T	F	T
Effect											
Return value =1	F	F	F	F	F		F	F			
Return value = 2	F	T	F	T	F		T	F			
Return value = 3	T	F	F	F	F		F	F			

(1,1,n) with n > 1 and return = 2 is unfeasible

Combining causes and effect

- One side =1 can result in return value = 1 (SCA), 2 (ISO) or all false

Causes	Rules													
Side1 = 1	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Side1 > 1	F	F	F	F	F	F	T	T	T	T	T	T	T	T
Side2 = 1	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Side2 > 1	F	F	T	T	T	T	F	F	F	F	T	T	T	T
Side3 = 1	T	F	T	F	F	F	T	F	F	F	T	T	T	F
Side3 > 1	F	T	F	T	T	T	F	T	T	T	F	F	F	T
Effect														
Return value =1	F	F	F	T	F	F	F	T	F	F	T	F	F	
Return value = 2	F	F	F	F	T	F	F	F	T	F	F	T	F	
Return value = 3	T	F	F	F	F	F	F	F	F	F	F	F	F	

Combining causes and effect

- No sides =1 can result in return value = 1 (SCA), 2 (ISO), 3 (EQI) or all false

Causes	Rules																
Side1 = 1	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F
Side1 > 1	F	F	F	F	F	F	T	T	T	T	T	T	T	T	T	T	T
Side2 = 1	T	T	F	F	F	F	T	T	T	T	F	F	F	F	F	F	F
Side2 > 1	F	F	T	T	T	T	F	F	F	F	T	T	T	T	T	T	T
Side3 = 1	T	F	T	F	F	F	T	F	F	F	T	T	T	F	F	F	F
Side3 > 1	F	T	F	T	T	T	F	T	T	T	F	F	F	T	T	T	T
Effect																	
Return value =1	F	F	F	T	F	F	F	T	F	F	T	F	F	T	F	F	F
Return value = 2	F	F	F	F	T	F	F	F	T	F	F	T	F	F	T	F	F
Return value = 3	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	F

Remove duplicate tests done previously

…and find test values for each rule (expected result is given by the effect). Max 17 tests

Causes	Rules																
Side1 = 1	T	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F	F
Side1 > 1	F	F	F	F	F	F	T	T	T	T	T	T	T	T	T	T	T
Side2 = 1	T	T	F	F	F	F	T	T	T	T	F	F	F	F	F	F	F
Side2 > 1	F	F	T	T	T	T	F	F	F	F	T	T	T	T	T	T	T
Side3 = 1	T	F	T	F	F	F	T	F	F	F	T	T	T	F	F	F	F
Side3 > 1	F	T	F	T	T	T	F	T	T	T	F	F	F	T	T	T	T
Effect																	
Return value =1	F	F	F	T	F	F	F	T	F	F	T	F	F	T	F	F	F
Return value = 2	F	F	F	F	T	F	F	F	T	F	F	T	F	F	T	F	F
Return value = 3	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	T	F

Decision table - Evaluations

- Strengths
  - Decision tables exercises combination of test values and expected results
  - Expected results are part of the process

- Weaknesses
  - Decision table tend to be very large (use pair/combination of causes to mitigate growth)
  - Filling the table costs effort (good knowledge of specification) and time