

Module 4 – Topic 4.2 –Lesson 4.2.3

Black Box Testing

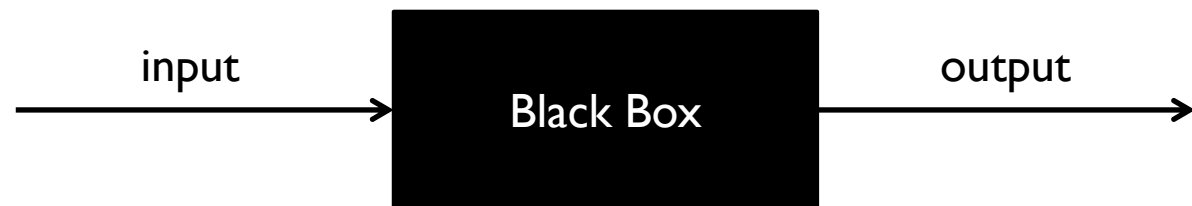
Lesson Outline

- ▶ **Boundary value testing**
 - ▶ Work well for independent, bounded physical quantities
 - ▶ Four types – normal vs. robust, weak vs. strong
- ▶ **Equivalence class testing**
 - ▶ Define an equivalence relation to partition input domain
 - ▶ Four types – normal vs. robust, weak vs. strong
- ▶ **Decision table testing**
 - ▶ Logical relationships among input variables (conditions)
 - ▶ Cause-effect relationships between conditions and actions
- ▶ **Guidelines and observations**



Black Box Testing


- ▶ Testing **behaviour** of the software
 - ▶ **Functional testing** (our focus)
 - ▶ Non-functional testing, e.g., load testing, usability testing
- ▶ Testing against a known specification, no knowledge of the internal structure



Recall The Impossibility of Test Everything

Specification: Take an integer input value, add 1 to input, divide it by 30000 and return the value

You do not see
implementation



Assume input value is a short int, which value(s) do you choose to test the implementation?

short int – 2^{16} values
int – 2^{32} values
long int – 2^{64} values

Input	Expected Output	Actual Output
0	0	
1	0	
-42	0	
32000	1	

Black box testing

- Boundary Value Testing
- Equivalence Class Testing
- Decision table testing



Functional Testing

- ▶ Use knowledge of the functional nature of a program to identify test cases for the program
 - ▶ Boundary value testing
 - ▶ Equivalence class testing
 - ▶ Decision table testing



- View a program as a function, input as [domain](#), output as [range](#)
- Design test cases based on boundary values, equivalence classes, business rules of input domain and output range



The General Black Box Testing Process

1. The specification are analysed
 - ▶ Boundary values, equivalence classes, decision table
2. Inputs are chosen based on the specification
 - ▶ Valid inputs are chosen to determine that the software under test (SUT) processes them correctly.
 - ▶ Invalid inputs may also need to be chosen to verify that the SUT detects them and handles them properly.
3. Expected outputs for those inputs are determined
4. Tests are constructed with the selected inputs
5. Tests are run
6. Actual outputs are compared with the expected outputs
7. A determination is made as to the proper functioning of the SUT








Boundary Value Testing

Rationale and Key Idea

- ▶ Usually focus on the input domain
- ▶ Derive boundary values from the extrema of independent, bounded physical quantities
- ▶ Because errors tend to occur near the boundary values for an input variable



Independent Bounded Physical Quantities

- ▶ A variable supports an **ordering relation** (usually some discrete values with **lower** and **upper** bounds) 
 - Mortgage application: income range [8000, 50000]; # of mortgage [0, 5]
 - Room environment control: Humidity [10, 45]; Temperature [-10, 25]
 - Triangle problem: edge length [1, MAX_INT] or [1, 200] (assume maximum allowed length)
 - NextDate: Month Jan → 1, Feb → 2 ... December → 12; Day [1, 31]; Year [1816, 2317]
- ▶ **Physical** quantities as opposed to logical variables 
 - six digits PINs or telephone numbers whose boundary values (000000, 999999) have little meaning
- ▶ **Independent**, as no consideration of the nature of the function, nor of the semantic meaning of variables 
 - Work well for incoming range and # of mortgage, humidity and temperature
 - No consideration of February, leap years, the year 2000, and the dependencies between Month, Day, Year in NextDate problem
 - No consideration of triangle properties, like $a+b \leq c$, $a \neq b \neq c$, etc.



Valid & Invalid Values

- ▶ Only test valid values (for each variable, take **5 values**)

Just below or above depends on the unit of the value

- ▶ minimum
- ▶ just above minimum
- ▶ a nominal value (e.g., average)
- ▶ just below maximum
- ▶ maximum

	min	min+	nom	max-	max
Edge Length	1	2	100	199	200
Month	Jan	Feb	Jun	Nov	Dec
Alphabet	a	b	m	y	z

- ▶ Also test invalid values for exception handling (for each variable, take **2 more values**)

- ▶ just below minimum
- ▶ just above maximum

	min-	max+
Edge Length	0	201
Month	-1	13
Alphabet	If strongly typed, N/A	

Single Fault Assumption or Not

- ▶ Single fault assumption – faults are due to incorrect values of a single variable
 - ▶ Holding the values of all but one variable at their nominal values, and letting that variable assume its full set of test values
- ▶ Or not – we are concerned with interaction among two or more variables
 - ▶ Taking the Cartesian product of the sets of test values of all variables



Four Types of Boundary Value Testing

		Single Fault Assumption	
		Yes	No
Invalid Values Tested	No	Weak Normal	Strong Normal
	Yes	Weak Robust	Strong Robust

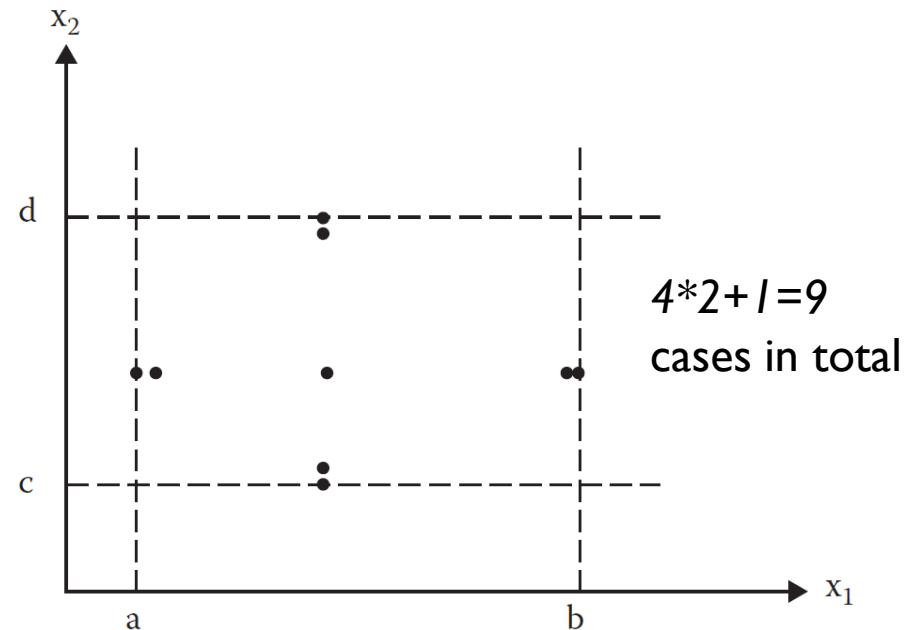
- The two dimensions are independent
- Robust generates more test cases than normal
- Strong generates more test cases than weak



Weak Normal Boundary Value Testing

- ▶ Or just normal boundary value testing
- ▶ Normal – test only valid values
 - ▶ min, min+, nom, max-, max
- ▶ Weak – single fault assumption
 - ▶ all but one variable as nom, and that variable assumes test values
- ▶ For n variables, $4n+1$ cases in total

Example: $a \leq x_1 \leq b$; $c \leq x_2 \leq d$



$(x_{1nom}, x_{2min}), (x_{1nom}, x_{2min+}), (x_{1nom}, x_{2nom}),$
 $(x_{1nom}, x_{2max-}), (x_{1nom}, x_{2max})$
 $(x_{1min}, x_{2nom}), (x_{1min+}, x_{2nom}), (x_{1max-}, x_{2nom}),$
 (x_{1max}, x_{2nom})

Weak Normal BVT – Mortgage Application

► Mortgage application

- income range [8000, 50000]: 8000, 8001, 29000, 49999, 50000
- # mortgage [0, 5]: 0, 1, 2, 4, 5

Case#	Income	# mortgage	Expected Output
1	29000	0	Approval
2	29000	1	Approval
3	29000	2	Approval
4	29000	4	Approval
5	29000	5	Approval
6	8000	2	Approval
7	8001	2	Approval
8	29000	2	Approval
9	49999	2	Approval
10	50000	2	Approval

Work well when the program is a function of **independent** variables that represent **bounded physical** quantities



Weak Normal BVT – Triangle Problem

- Assume edge length [1, 200] – 1, 2, 100, 199, 200

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	100	100	Equilateral
9	100	199	100	Isosceles
10	100	200	100	Not a triangle
11	1	100	100	Isosceles
12	2	100	100	Isosceles
13	100	100	100	Equilateral
14	199	100	100	Isosceles
15	200	100	100	Not a triangle

Do you find it

- redundant testing?
- untested functionality?

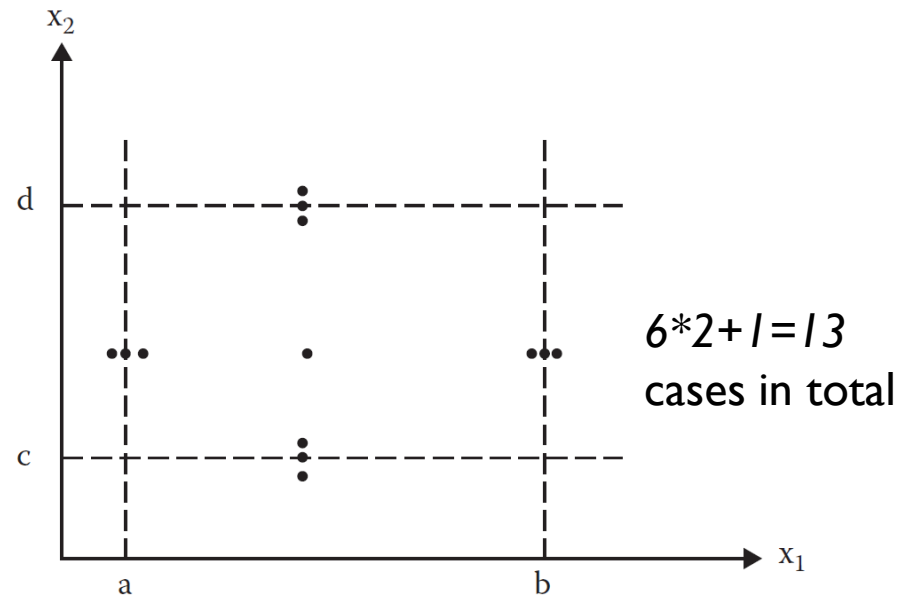
This is due to no consideration of triangle semantics, $a+b \leq c$, $a \neq b \neq c$

Kind of accidental. If nominal value takes 101, there will be no test cases for “Not a triangle”

Weak Robust Boundary Value Testing

- ▶ Or just robust boundary value testing
- ▶ Purpose: **exception handling**
 - ▶ may cause runtime errors in strongly typed languages
- ▶ Robust - test valid and **invalid** values
 - ▶ **min-**, min, min+, nom, max-, max, **max+**
- ▶ Weak – single fault assumption
 - ▶ all but one variable as nom, that variable assumes test values
- ▶ For n variables, $6n+1$ cases in total

Example: $a \leq x_1 \leq b$; $c \leq x_2 \leq d$



$(x_{l\text{nom}}, x_{2\text{min-}}), (x_{l\text{nom}}, x_{2\text{min}}), (x_{l\text{nom}}, x_{2\text{min+}}),$
 $(x_{l\text{nom}}, x_{2\text{nom}}), (x_{l\text{nom}}, x_{2\text{max-}}), (x_{l\text{nom}}, x_{2\text{max}}),$
 $(x_{l\text{nom}}, x_{2\text{max+}})$
 $(x_{l\text{min-}}, x_{2\text{nom}}), (x_{l\text{min}}, x_{2\text{nom}}), (x_{l\text{min+}}, x_{2\text{nom}}),$
 $(x_{l\text{max-}}, x_{2\text{nom}}), (x_{l\text{max}}, x_{2\text{nom}}), (x_{l\text{max+}}, x_{2\text{nom}})$

Weak Robust BVT – Mortgage Application

► Mortgage application

- income range [8000, 50000]: 7999, 8000, 8001, 29000, 49999, 50000, 50001
- # mortgage [0, 5]: -1, 0, 1, 2, 4, 5, 6

Case#	Income	# mortgage	Expected Output
1	29000	-1	# mortgage should not be negative
2	29000	6	Reject. Too many mortgage
3	7999	2	Reject. Income is low.
4	50001	2	Reject. Too rich. No need mortgage.

Weak normal test cases (slide 14) are omitted for clarify



Weak Robust BVT – Triangle Problem

- ▶ Assume edge length $[1, 200]$ – 0, 1, 2, 100, 199, 200, 201

Case#	a	b	c	Expected Output
1	100	100	0	Value of c is not in the permitted range [1,200]
2	100	100	201	Value of c is not in the permitted range [1,200]
3	100	0	100	Value of b is not in the permitted range [1,200]
4	100	201	100	Value of b is not in the permitted range [1,200]
5	0	100	100	Value of a is not in the permitted range [1,200]
6	201	100	100	Value of a is not in the permitted range [1,200]

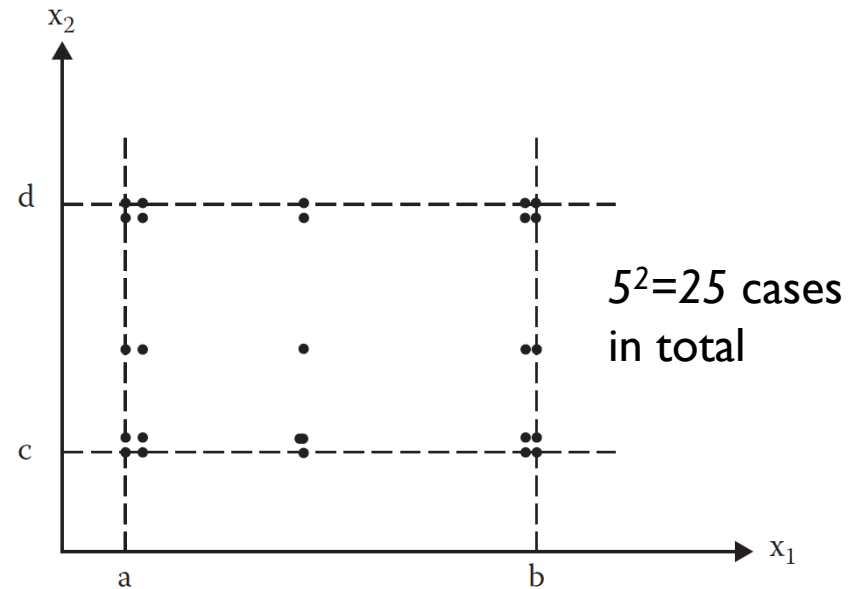
Weak normal test cases (slide 15) are omitted for clarify



Strong Normal Boundary Value Testing

- ▶ Also known as worst case boundary value testing
- ▶ Normal – test only valid values
 - ▶ min, min+, nom, max-, max
- ▶ Strong – multiple fault assumption
 - ▶ The Cartesian product of five test values of all variables
- ▶ For n variables, 5^n cases in total
 - ▶ More thorough, but much more effort
 - ▶ Applicable when failure is extremely costly

Example: $a \leq x_1 \leq b$; $c \leq x_2 \leq d$



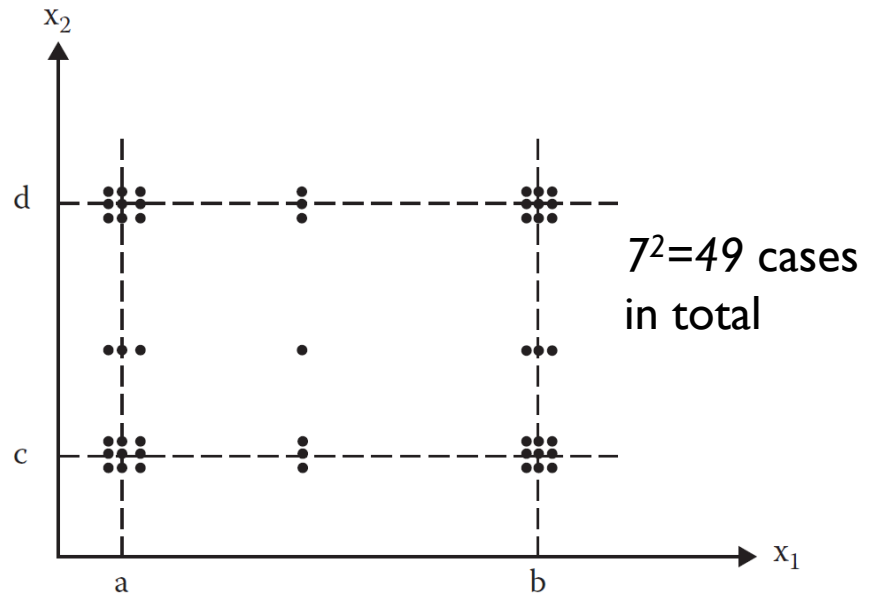
Weak normal BVT: $4n+1$

Weak robust BVT: $6n+1$

Strong Robust Boundary Value Testing

- ▶ Also known as robust worst case boundary value testing
- ▶ Robust – test valid and **invalid** values
 - ▶ **min-**, min, min+, nom, max-, max, **max+**
- ▶ Strong – multiple fault assumption
 - ▶ The **Cartesian product** of **seven** test values of all variables
- ▶ For n variables, 7^n cases in total

Example: $a \leq x_1 \leq b$; $c \leq x_2 \leq d$



Strong Normal BVT – Mortgage Application

► Mortgage application

- income range [8000, 50000]: 8000, 8001, 29000, 49999, 50000
- # mortgage [0, 5]: 0, 1, 2, 4, 5

Case#	Income	# mortgage	Expected Output
1	8000	0	Approval
2	8000	1	Approval
3	8000	2	Approval
4	8000	4	Approval
5	8000	5	Approval
6	8001	0	Approval
7	8001	1	Approval
8	8001	2	Approval
9	8001	4	Approval
10	8001	5	Approval
...
25	50000	5	Approval

25 combinations of every test value of income range and every test value of # mortgage



Strong Robust BVT – Mortgage Application

► Mortgage application

- income range [8000, 50000]: 7999, 8000, 8001, 29000, 49999, 50000, 50001
- # mortgage [0, 5]: -1, 0, 1, 2, 4, 5, 6

Case#	Income	# mortgage	Expected Output
1	7999	-1	# mortgage should not be negative
2	7999	0	Reject. Income is low.
3	7999	1	Reject. Income is low.
4	7999	2	Reject. Income is low.
5	7999	4	Reject. Income is low.
6	7999	5	Reject. Income is low.
7	7999	6	Reject. Income is low. Too many mortgage.
...

49 combinations of every test value of income range and every test value of # mortgage



Special Value Testing

- ▶ Intuitive, widely practiced, but the least uniform
- ▶ A tester uses domain knowledge, experience with similar programs, and information about “soft spots” to devise tests
 - ▶ Triangle problem: $a+b \leq c$, $a \neq b \neq c$
 - ▶ NextDate: Feb 28, Feb 29, leap years, the year 2000

Cons: ad hoc, subjective,
no guideline are used
other than “best
engineering judgment”

Pros: applies domain knowledge,
often more effective than the
test cases generated by
boundary value methods



Equivalence Class Testing

Motivation and Key Idea

▶ Motivation

- ▶ A sense of complete testing
- ▶ Avoid redundancy



▶ Key idea

- ▶ Partition the entire input set into disjoint subsets by an equivalence relation, the union of which is the entire set
- ▶ Identify test cases by using one value from each equivalence class

Coverage – a form of completeness
Disjointness – no redundancy



Equivalence Classes

- ▶ Find a good equivalence relation to partition
 - ▶ the values of an equivalence class are “treated the same way”
- ▶ Equivalence relation selection
 - ▶ Understand the domain
 - ▶ Knowledge of the specification (not the source code)
 - ▶ Knowledge of dependency among input variables



Equivalence Classes – Examples

Equivalence classes of Mortgage Application on input domain

Income range: **low income** [0,7999]; **approval income** [8000, 50000]; **too rich** ≥ 50001

mortgage: **invalid #** ≤ -1 ; **approval #** [0, 5]; **too many #** ≥ 6

Applicant type: **approval type** {personal}; **invalid type** {trust, company, partnership}

Property type: **approval type** {Apartment}; **invalid type** {House, TownHose, Land}

Note: Equivalence classes of income range and # mortgage are bounded quantities. But they are treated as sets in equivalence class testing. See the slide “Edge Testing” for a hybrid of BVT and ECT.



Equivalence Classes –Examples

Equivalence classes of NextDate on input domain

Month: **invalid** ≤ 0 ; **valid** $[1, 12]$; **invalid** ≥ 13

Day: **invalid** ≤ 0 ; **valid** $[1, 31]$; **invalid** ≥ 32

Year: **invalid** ≤ 1815 ; **valid** $[1816, 2317]$; **invalid** ≥ 2318

Are these equivalence classes very useful for testing?

The “treatment” is at the valid/invalid level. Can you choose some more useful equivalence relation?



Equivalence Classes –Examples

Equivalence classes of Triangle Problem (based on output range)

Equilateral: $\{ \langle a, b, c \rangle \mid \text{the triangle has three sides of equal length} \} = \{ \langle 5, 5, 5 \rangle, \langle 1, 1, 1 \rangle, \langle 100, 100, 100 \rangle, \dots \}$

Isosceles: $\{ \langle a, b, c \rangle \mid \text{the triangle has two sides of equal length} \} = \{ \langle 5, 5, 6 \rangle, \langle 4, 4, 2 \rangle, \langle 100, 100, 90 \rangle, \dots \}$

Scalene: $\{ \langle a, b, c \rangle \mid \text{the triangle has three unequal sides} \} = \{ \langle 5, 4, 6 \rangle, \langle 3, 4, 5 \rangle, \langle 80, 100, 90 \rangle, \dots \}$

Not a triangle: $\{ \langle a, b, c \rangle \mid \text{sides do not satisfy triangle property} \} = \{ \langle 5, 4, 1 \rangle, \langle 3, 4, 9 \rangle, \langle 100, 100, 200 \rangle, \dots \}$

Invalid side: $\{ \langle a, b, c \rangle \mid \text{side length} \leq 0 \text{ or side length} \geq 201 \} = \{ \langle 0, 4, 1 \rangle, \langle -1, -6, 9 \rangle, \langle -6, 201, 500 \rangle, \dots \}$

Can we define equivalence classes on input domain? Check Chapter 6 of Software Testing: A Craftsman's Approach.



Four Types of Equivalence Class Testing

		Single Fault Assumption	
		Yes	No
Invalid Values Tested	No	Weak Normal	Strong Normal
	Yes	Weak Robust	Strong Robust

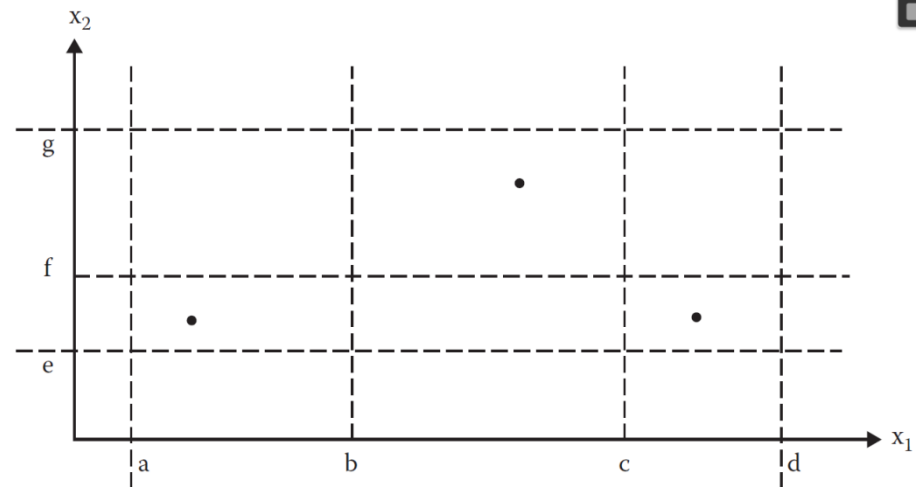
- The two dimensions are independent
- Robust generates more test cases than normal
- Strong generates more test cases than weak



Weak Normal Equivalence Class Testing

- ▶ Normal – test only valid values
- ▶ Weak – single fault assumption
 - ▶ one value for each equivalence class
- ▶ The number of test cases is the same as classes in the partition with the largest number of subsets

Example: $a \leq x_1 \leq d$, with intervals $[a,b)$, $[b,c)$, $[c,d]$; $e \leq x_2 \leq g$, with intervals $[e,f)$, $[f,g)$

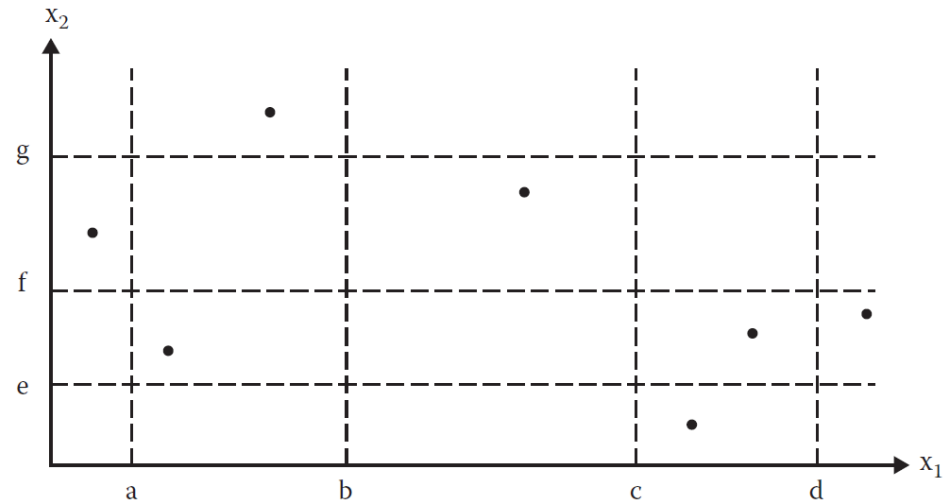


Note: Partition based on valid and invalid intervals. The ordering and bound of quantities are irrelevant to equivalence class testing.

Weak Robust Equivalence Class Testing

- ▶ Robust – test valid and **invalid** values
- ▶ Purpose: **exception handling**
 - ▶ may cause runtime errors in strongly typed languages
- ▶ Weak – single fault assumption
 - ▶ For valid inputs, one value for each equivalence class (same as weak normal ECT)
 - ▶ For invalid inputs, one invalid value and the remaining values will all be valid

Example: $a \leq x_1 \leq d$, with intervals $[a,b)$, $[b,c)$, $[c,d]$; $e \leq x_2 \leq g$, with intervals $[e,f)$, $[f,g)$

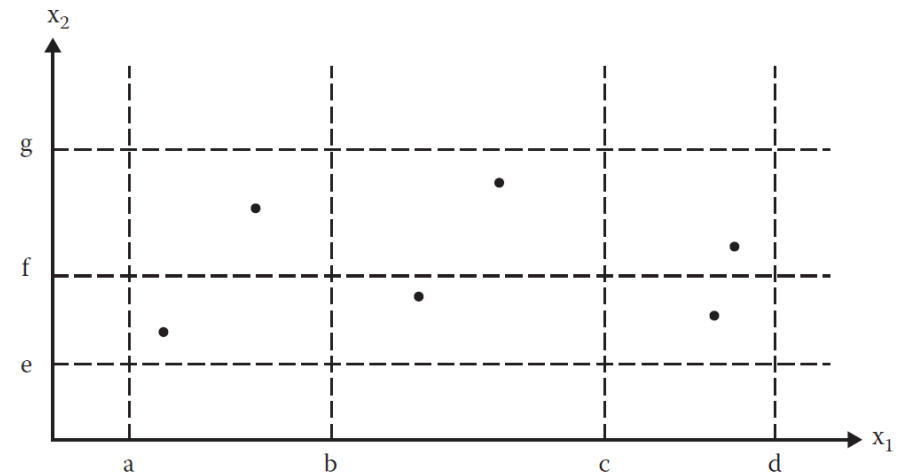


of test cases = # of weak normal ECT test cases + the sum of invalid equivalence classes of all the variables

Strong Normal Equivalence Class Testing

- ▶ Normal – test only valid values
- ▶ Strong – multiple fault assumption
 - ▶ the **Cartesian product** of the valid equivalence classes
- ▶ The notion of “completeness”
 - ▶ Cover all the equivalence classes
 - ▶ Each possible combination of inputs

Example: $a \leq x_1 \leq d$, with intervals $[a,b)$, $[b,c)$, $[c,d]$; $e \leq x_2 \leq g$, with intervals $[e,f)$, $[f,g)$

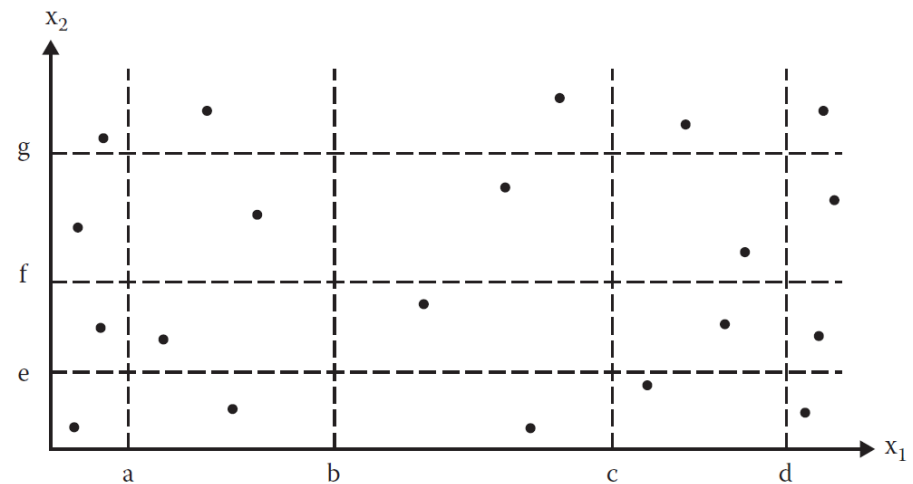


of test cases = the multiplication of the number of valid equivalence classes of all the variables

Strong Robust Equivalence Class Testing

- ▶ Robust – test valid and **invalid** values
- ▶ Strong – multiple fault assumption
 - ▶ the **Cartesian product** of all the equivalence classes
- ▶ The notion of “completeness”
 - ▶ Cover all the equivalence classes
 - ▶ Each possible combination of inputs

Example: $a \leq x_1 \leq d$, with intervals $[a,b)$, $[b,c)$, $[c,d]$; $e \leq x_2 \leq g$, with intervals $[e,f)$, $[f,g)$



of test cases = the multiplication of the number of all equivalence classes of all the variables

Weak (Strong) Normal ECT – Mortgage Application

Equivalence classes of Mortgage Application on input domain

Income range: **low income** $[0, 7999]$; **approval income** $[8000, 50000]$; **too rich** ≥ 50001

mortgage: **invalid #** ≤ -1 ; **approval #** $[0, 5]$; **too many #** ≥ 6

Applicant type: **approval type** {personal}; **invalid type** {trust, company, partnership}

Property type: **approval type** {Apartment}; **invalid type** {House, TownHose, Land}

case #	income	#mortgage	Applicant type	Property type	Expected Output
1	10001	3	personal	Apartment	Approval

Strong Normal ECT is identical to Weak Normal ECT,
because no valid subintervals of variables exist.



Weak Robust ECT – Mortgage Application

case #	income	#mortgage	Applicant type	Property type	Expected Output
1	10001	3	personal	Apartment	Approval
2	5000	3	personal	Apartment	Reject. Income is low.
3	85000	3	personal	Apartment	Reject. Too rich.
4	10001	-10	personal	Apartment	#mortgage not be negative
5	10001	8	personal	Apartment	Reject. Too many mortgages.
6	10001	3	trust	Apartment	Reject. Invalid applicant type
7	10001	3	personal	Land	Reject. Invalid property type

The first test case is for “Normal”, and the rest 6 test cases are for “Robust”.



Strong Robust ECT – Mortgage Application

case #	income	#mortgage	Applicant type	Property type	Expected Output
1	5000	-10	personal	Apartment	Reject ...
2	5000	-10	personal	Land	Reject ...
3	5000	-10	trust	Apartment	Reject ...
4	5000	-10	trust	Land	Reject ...
5	5000	3	personal	Apartment	Reject ...
6	5000	3	personal	Land	Reject ...
7	5000	3	trust	Apartment	Reject ...
8	5000	3	trust	Land	Reject ...
9	5000	8	personal	Apartment	Reject ...
10	5000	8	personal	Land	Reject ...
11	5000	8	trust	Apartment	Reject ...
12	5000	8	trust	Land	Reject ...
...

How many test cases? Hint: the Cartesian product of the number of all equivalence classes of the four variable.



Weak (Strong) Normal ECT – Triangle Problem

case #	a	b	c	Expected Output
1	5	5	5	Equilateral
2	2	2	3	Isosceles
3	3	4	5	Scalene
4	4	1	2	Not a triangle

Strong Normal ECT is identical to Weak Normal ECT, because no valid subsets of variables exist.



Weak Robust ECT – Triangle Problem

case #	a	b	c	Expected Output
1	-10	5	5	Value of a is not in the permitted range
2	250	5	5	Value of a is not in the permitted range
3	5	-10	5	Value of b is not in the permitted range
4	5	250	5	Value of b is not in the permitted range
5	5	5	-10	Value of c is not in the permitted range
6	5	5	250	Value of c is not in the permitted range

Weak normal ECT test cases are omitted for clarity



Strong Robust ECT – Triangle Problem

case #	a	b	c	Expected Output
1	-10	5	5	Value of a is not in the permitted range
2	-10	5	-10	Value of a, c is not in the permitted range
3	-10	5	250	Value of a, c is not in the permitted range
4	-10	-10	5	Value of a, b is not in the permitted range
5	-10	-10	-10	Value of a, b, c is not in the permitted range
6	-10	-10	250	Value of a, b, c is not in the permitted range
7	-10	250	5	Value of a, b is not in the permitted range
8	-10	250	-10	Value of a, b, c is not in the permitted range
9	-10	250	250	Value of a, b, c is not in the permitted range
...

How many test cases?



Weak Robust ECT – NextDate

Equivalence classes of NextDate on input domain

Month: **invalid** ≤ 0 ; **valid** $[1, 12]$; **invalid** ≥ 13

Day: **invalid** ≤ 0 ; **valid** $[1, 31]$; **invalid** ≥ 32

Year: **invalid** ≤ 1815 ; **valid** $[1816, 2317]$; **invalid** ≥ 2318

Partition based on valid and invalid intervals.

case #	Month	Day	Year	Expected Output
1	6	15	1910	16/06/1910 (16 June 1910)
2	-2	15	1910	Value of month not in $[1, 12]$
3	17	15	1910	Value of month not in $[1, 12]$
4	6	-1	1910	Value of day not in $[1, 31]$
5	6	35	1910	Value of day not in $[1, 31]$
6	6	15	1811	Value of year not in $[1816, 2317]$
7	6	15	2335	Value of year not in $[1816, 2317]$

Better Equivalence Classes – NextDate

Equivalence classes of NextDate on input domain

Month: M1={month has 30 days}, M2={month has 31 days}, M3={month is February}

Day: D1={ $1 \leq \text{day} \leq 28$ }, D2={day=29}, D3={day=30}, D4={day=31}

Year: Y1={year=2000}, Y2={a non-century leap year}, Y3={a common year}

Weak Normal ECT

case #	Month	Day	Year	Expected Output
1	6	15	2000	16/06/2000 (16 June 2000)
2	7	29	1996	30/07/1996 (30 July 1996)
3	2	30	2002	Invalid input date
4	6	31	2000	Invalid input date

Mechanical selection of input values makes no consideration of our domain knowledge, thus the two impossible dates.

Edge Testing (A Hybrid of BVT and ECT)

- ▶ Applies when contiguous ranges of a variable constitute equivalence classes

each boundary randomly
choose 3 values (value
below boundary, value
equals boundary, value
larger than boundary)

- ▶ Mortgage application: income range, # of mortgage
- ▶ Room temperature control: humidity, temperature
- ▶ Edge testing tests the boundaries of the classes

Example: $a \leq x_1 \leq d$, with intervals $[a,b)$, $[b,c)$, $[c,d]$; $e \leq x_2 \leq g$, with intervals $[e,f)$, $[f,g]$

Normal test values for x_1 : $\{a, a+, b-, b, b+, c-, c, c+, d-, d\}$

Robust test values for x_1 : $\{a-, a, a+, b-, b, b+, c-, c, c+, d-, d, d+\}$

Normal test values for x_2 : $\{e, e+, f-, f, f+, g-, g\}$

Robust test values for x_2 : $\{e-, e, e+, f-, f, f+, g-, g, g+\}$

Do not include the nominal
values that we had with
boundary value testing, because
they will be covered by ECT.

- ▶ Once the sets of edge values are determined, edge testing can follow any of the four types of equivalence class testing.
-





Decision Table Testing

Motivation and Applicability

- ▶ **For modelling complicated logic**
 - ▶ Lots of decision making
 - ▶ Logical relationships among input variables (conditions)
 - ▶ Cause-effect relationships between conditions and actions
 - ▶ Evaluation involving subsets of conditions
- ▶ **Applicable when**
 - ▶ Ordering of condition evaluation is irrelevant
 - ▶ Ordering of rule evaluation is irrelevant
 - ▶ Only one rule can evaluate to true at once
 - ▶ Ordering of action execution is irrelevant



Decision Table Testing – How to's

▶ How-to

- ▶ Determine conditions and values
- ▶ Determine actions
- ▶ Encode rules
- ▶ Encode actions for rules
- ▶ Verify rules
- ▶ Simplify rules

▶ Possible values for conditions

- ▶ Booleans, don't care (-), extended values

- ▶ If all the conditions are binary, Limited Entry Decision Table. Otherwise, Extended Entry Decision Table

	Rule1	Rule2	Rule3
C1	T	T	T
C2	T	T	F
C3	T	F	-
A1	X		
A2	X		X
A3		X	

▶ Test case design

- ▶ Conditions as inputs
- ▶ Actions as outputs
- ▶ Rules as test cases



Decision Table for Triangle Problem

	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1: a, b, c form a triangle?	F	T	T	T	T	T	T	T	T
C2: a = b?	-	T	T	T	T	F	F	F	F
C3: a = c?	-	T	T	F	F	T	T	F	F
C4: b = c?	-	T	F	T	F	T	F	T	F
A1: not a triangle	X								
A2: scalene									X
A3: isosceles					X		X	X	
A4: equilateral		X							
A5: impossible			X	X		X			

Can you expand C1 to more fine-grained conditions?

Hint: $a < b+c$, $b < a+c$, $c < a+b$



Revised Equivalence Classes of NextDate

Year:

$Y1 = \{\text{a leap year}\}$

$Y2 = \{\text{a common year}\}$

Month:

$M1 = \{\text{month has 30 days}\}$

$M2 = \{\text{month has 31 days except December}\}$

$M3 = \{\text{month is December}\}$

$M4 = \{\text{month is February}\}$

Day:

$D1 = \{1 \leq \text{day} \leq 27\}$

$D2 = \{\text{day} = 28\}$

$D3 = \{\text{day} = 29\}$

$D4 = \{\text{day} = 30\}$

$D5 = \{\text{day} = 31\}$

The Cartesian product of Year,
Month, Day contains 40 elements



Partial Decision Table for NextDate

	R1	R2	R3	R4	R5
C1: day in	D1, D2, D3	D4	D5	D1,D2,D3,D4	D5
C2: month in	M1	M1	M1	M2	M2
C3: year in	-	-	-	-	-
A1: impossible			X		
A2: day+1	X			X	
A3: reset day		X			X
A4: month+1		X			X
A5: reset month					
A6: year+1					

Note that we use equivalence classes as values for conditions. This type of decision table is called Extended Entry Decision Table.

If we use Boolean values for conditions like “day in D1”, “month in M1”, “year in Y1”, etc., how many rules will we have in the decision table?
Hint: conditions will be mutually exclusive.

Guidelines and Considerations

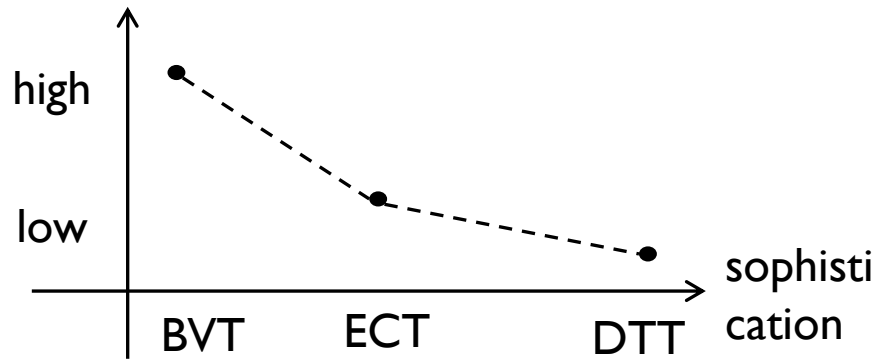
BVT, ECT, DTT

- ▶ Commonality: treat the program as a function that maps inputs to outputs
- ▶ Testing effort and efficiency?

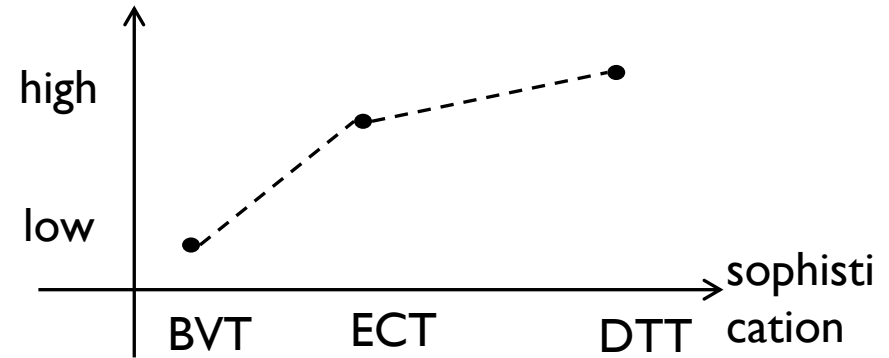


Test Effort

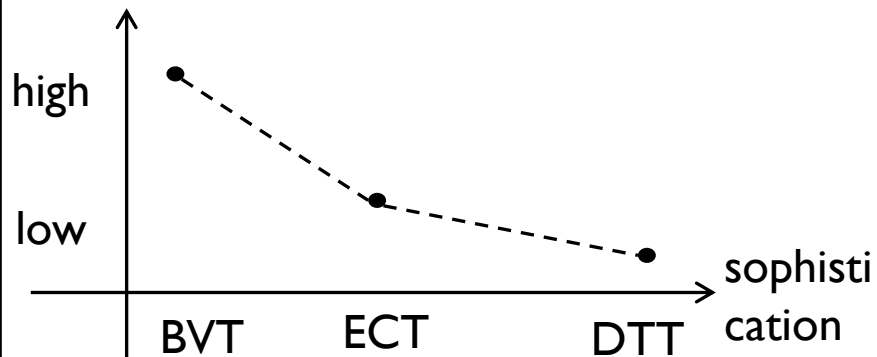
Number of test cases



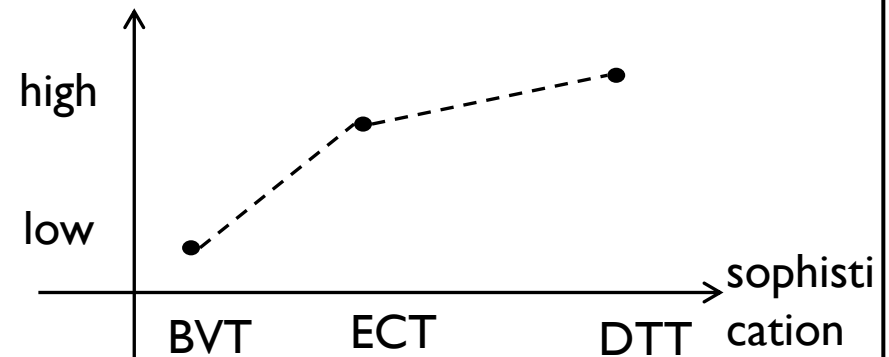
Effort to identify test cases



Redundancy



Completeness



Testing Effort – An Analysis

- ▶ BVT – no recognition of data or logical dependencies
 - ▶ Mechanical, easy to automate
- ▶ ECT – need to consider data dependencies and program logics
 - ▶ Defining the equivalence relation
- ▶ DTT – need to consider logical relationships among input variables as well as cause-effect relationships between conditions and actions
 - ▶ Do not scale up very well (especially Limited Entry Decision Table in which all the conditions are binary (true/false))



Testing Efficiency

▶ Pros

- ▶ Directs the tester to a very small subset of test inputs that are both efficient and effective in finding the bugs

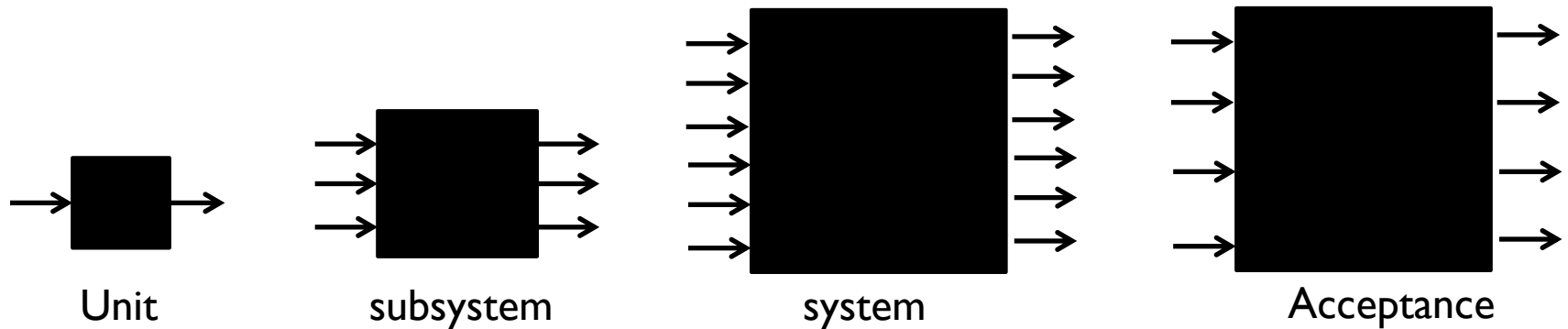
▶ Cons

- ▶ Cannot know how much of the implementation have been tested
- ▶ Have some criteria like untested functionality, test redundancy, but no notion of testing coverage like in white box testing



Applicability

- ▶ Black box testing can be applied at all levels of system development



Functional Testing Technique Selection

- ▶ **Explore program attributes**
 - ▶ Whether the variables represent physical (P) or logical (L) quantities
 - ▶ Whether dependencies exist among variables
 - ▶ Whether single or multiple fault are assumed
 - ▶ Whether exception handling is prominent



Decision Table for Technique Selection

C1: Physical (P) or Logical (L)	P	P	P	P	P	L	L	L	L	L
C2: Independent variable?	T	T	T	T	F	T	T	T	T	F
C3: Single fault?	T	T	F	F	-	T	T	F	F	-
C4: Exception handling?	T	F	T	F	-	T	F	T	F	-
A1:Weak normal BVT		X								
A2:Weak robust BVT	X									
A3: Strong normal BVT				X						
A4: Strong robust BVT			X							
A5:Weak normal ECT		X					X			
A6:Weak robust ECT	X					X				
A7: Strong normal ECT				X					X	
A8: Strong robust ECT			X		X			X		X
A9: Decision table					X					X

