Software Engineering
Lecture 05

Software Testing

Referenced documents may be accessed via the URLs located on the course Angel page. Off-campus access will require authentication.

Outline

- Functional Testing
  - Methods
    - Boundary Value Analysis
    - Equivalence Class Testing
    - Decision Tables
  - Guidelines for Functional Testing
- Structural Testing

Functional Testing

Boundary Value Analysis

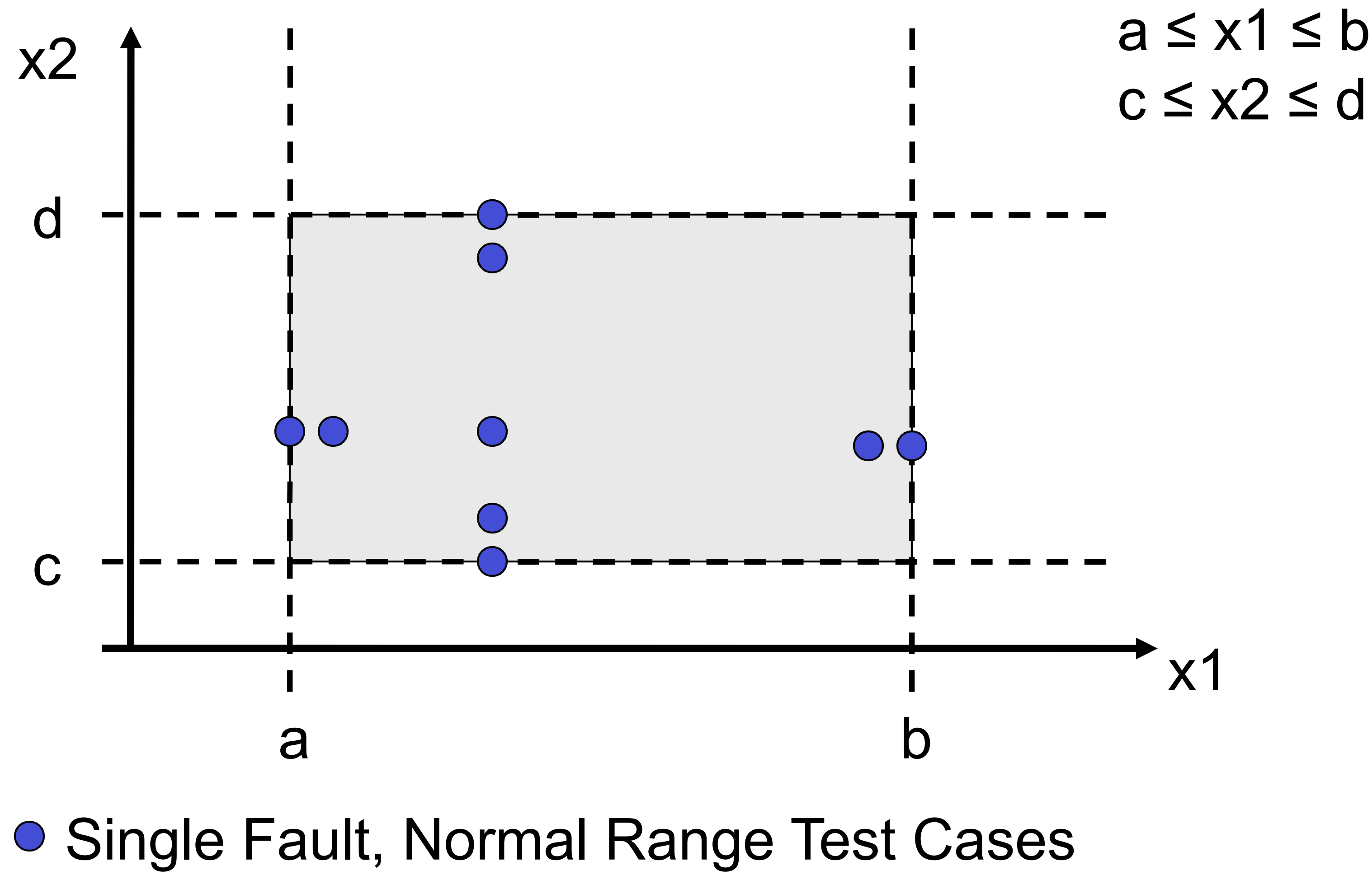
- BVA focuses on testing boundaries of the input space
- Assumes errors tend to occur at the extremes of the input values
- Many variations of BVA

Boundary Value Analysis

- "Single Fault" Assumption
  - Failures are rarely the result of two or more faults
  - Test cases selected by examining the extremes of one variable's values while holding all remaining variable values at their nominal values
- For a single input variable ***x1*** with a fixed range, select the following set of tests of valid inputs

  {Min, Min+, Nom, Max-, Max}
  - Minimum Value (i.e. Min)
  - Just Above Minimum Value (i.e. Min+)
  - Nominal Value (i.e. Nom)
  - Just Below Maximum Value (i.e. Max-)
  - Maximum Value (i.e. Max)
- For ***n*** variables, ***4n+1*** test cases selected

BVA with Two Variables



n=2 yields 4*n+1 = 9 test cases

BVA with Robustness Testing

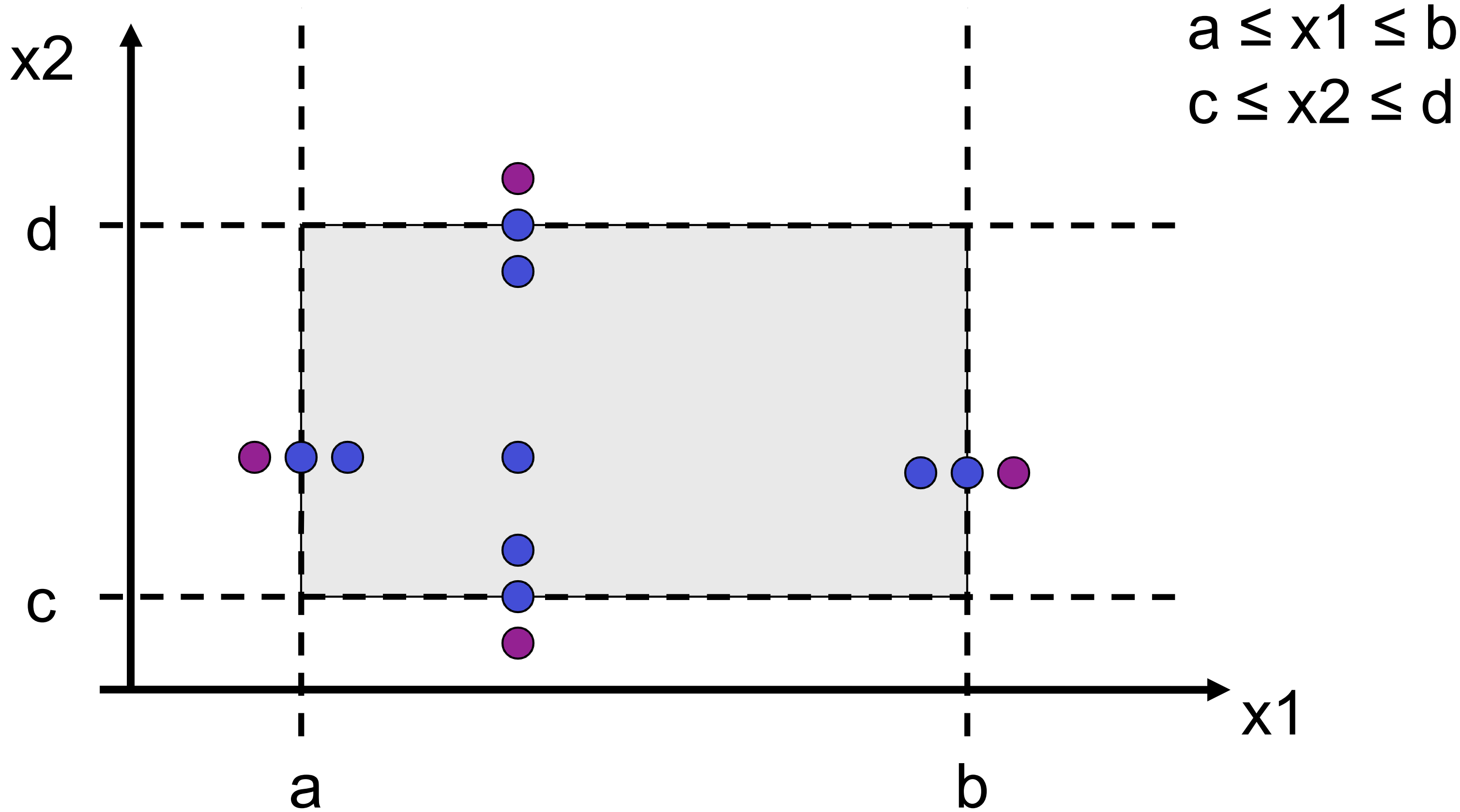
- Still “Single Fault” Assumption
- Select the set of five test cases identified for a variable under BVA

  {Min, Min+, Nom, Max-, Max}

  and

  add test values just exceeding the minimum and maximums {Min-, Max+}

BVA with Robustness Testing



- Single Fault, Normal Range Test Cases
- Additional Robustness Test Cases

n=2 yields 13 test cases

BVA with Worst-Case Testing

- Disregard “single-fault” assumption

- Start with the five test cases identified for a variable under BVA

  {Min, Min+, Nom, Max-, Max}

  and

  augment with ***Cartesian products*** of these sets

BVA with Worst-Case Testing

- X1 with {Min1, Min1+, Nom1, Max1-, Max1} and
  X2 with {Min2, Min2+, Nom2, Max2-, Max2}

Test Cases are the Cartesian Product of the two sets

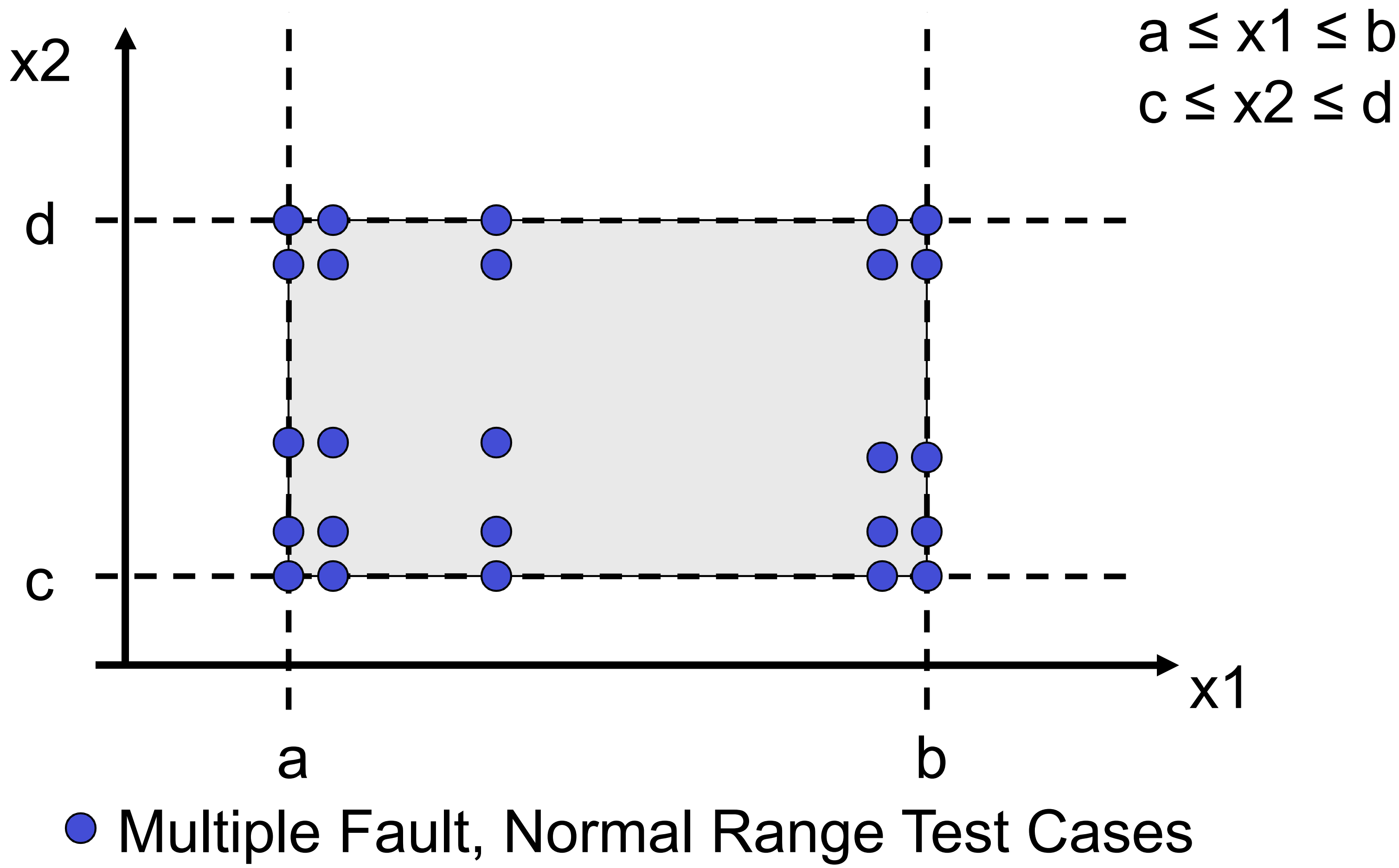
$X1 \times X2 =$

$\{<x1,x2> :$

$x1 \in$ {Min1, Min1+, Nom1, Max1-, Max1} X

$x2 \in$ {Min2, Min2+, Nom2, Max2-, Max2} }

BVA with Worst-Case Testing



n=2 yields 25 test cases

BVA Robust Worst-Case Testing

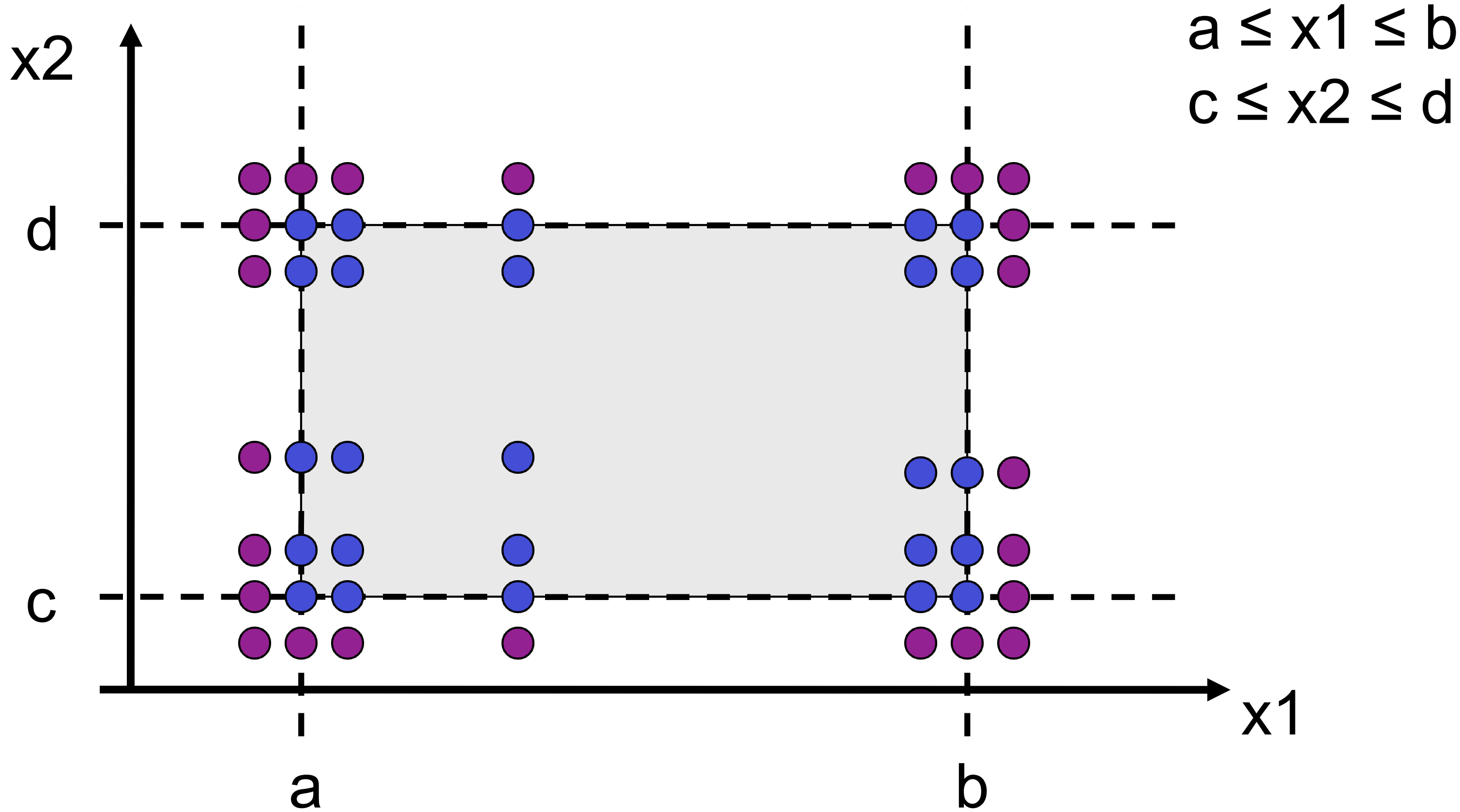
- Select the seven test cases identified for a variable under robust BVA

  {Min-, Min, Min+, Nom, Max-, Max, Max+}

  and

  augment with ***Cartesian products*** of these sets

BVA Robust Worst-Case Testing



- Multiple Fault, Normal Range Test Cases
- Additional Robustness Test Cases

n=2 yields 49 test cases

Comparison of Two-Variable BVA Strategies

# of Tests Generated as a Function of Fault Model and Robustness Criteria		Fault Model	
		Single Fault	Multiple Fault
Robustness Criteria	Normal Values	9	25
	Normal and Abnormal Values	13	49

Strengths and Limitations of BVA

- Can be applied on ***input*** or ***outputs***
- Works well for problems where
  - Variables are ***independent*** of each other
  - Variables represent physical quantities
    - Examples: pressure, temperature, humidity, etc.
    - Examples of “logical” quantities: PIN numbers, telephone numbers, etc.
- BVA can be automated when program specifications have been formally specified

Equivalence Class Testing

- Partition the entire set of ***input*** or ***output*** values into distinct subsets as a function of the problem
  - Partitioning entire set addresses <u>completeness</u>
  - Distinct subsets prevents <u>redundancy</u>
- All elements within a subset should have some factor in common such that all elements are equivalent from a testing point of view
- Tests are determined by selecting one test from each subset

Equivalence Class Testing

- Single versus multiple fault assumption
  - ***Weak*** refers to Single Fault assumption
  - ***Strong*** refers to Multiple Fault assumption
- Valid versus invalid data
  - ***Normal*** refers to valid data only
  - ***Robust*** refers to both valid and invalid data

- In the example, we assume two variables

  $a \leq x1 \leq d$ with intervals [a,b), [b,c), [c,d]

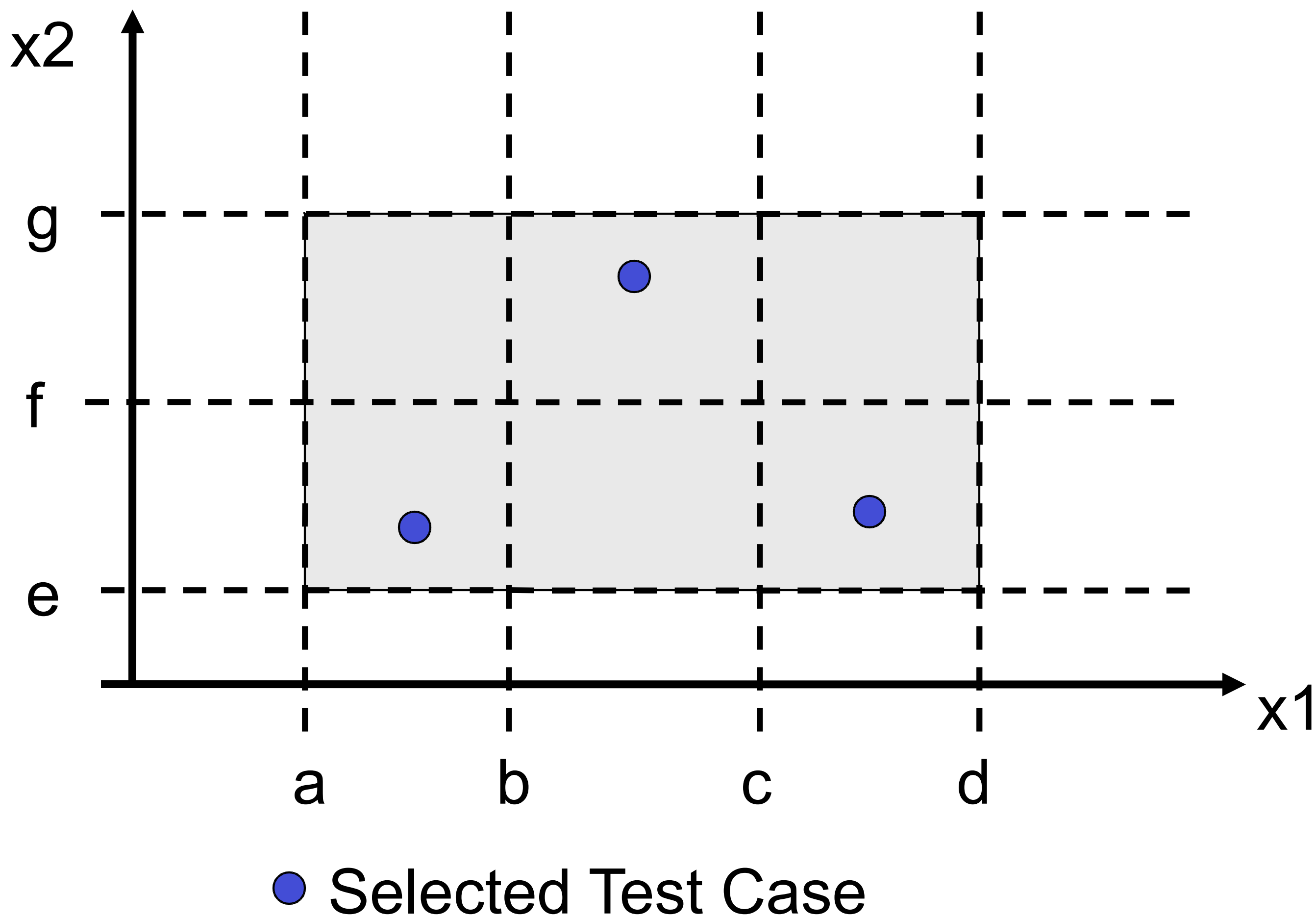
  $e \leq x2 \leq g$ with intervals [e,f), [f,g]

Weak Normal Equivalence Class Testing

- Use one value from each equivalence class

- Same number of test cases as largest number of variable value subsets

Weak Normal Equivalence Class Testing

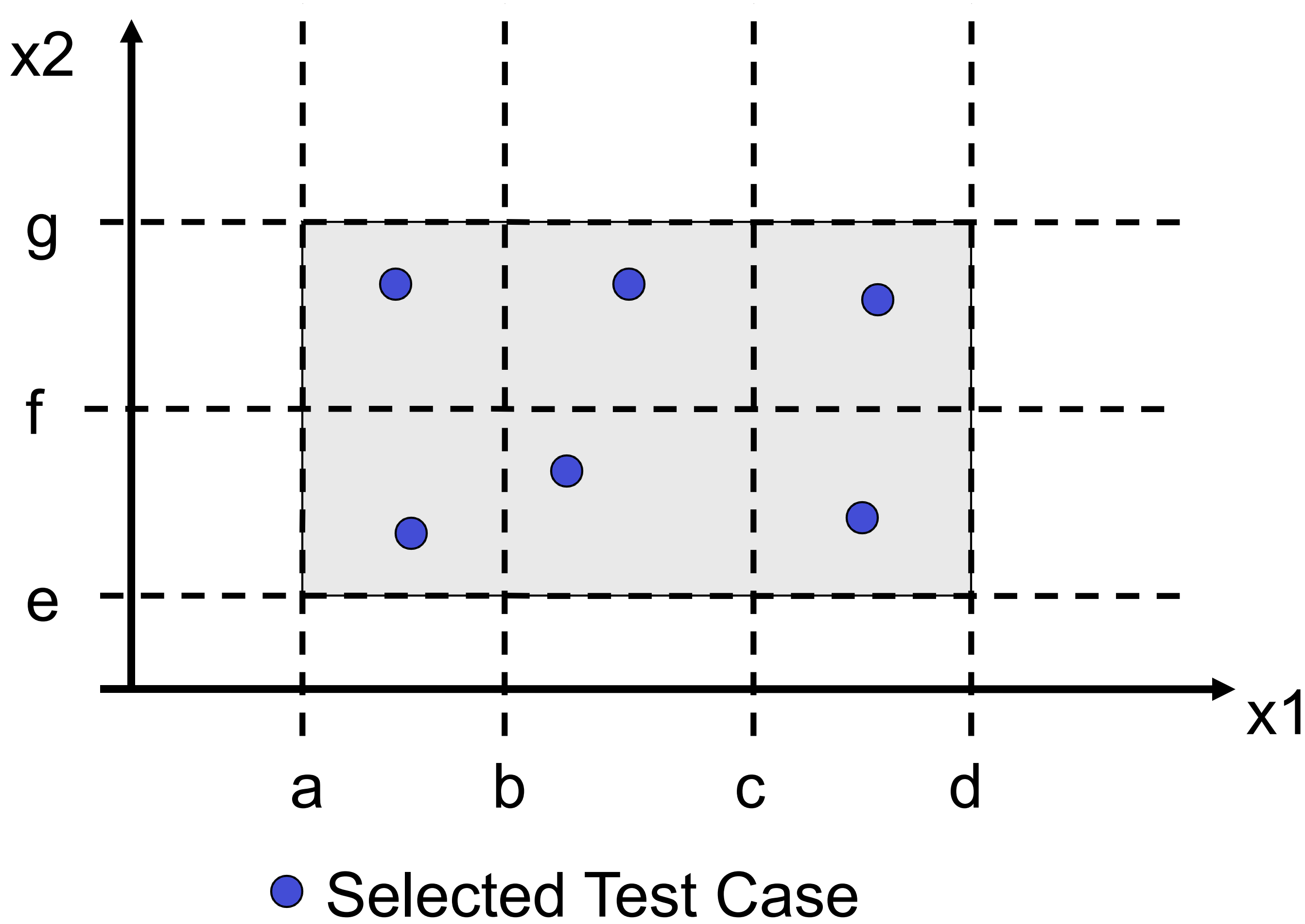


3 test cases

Strong Normal Equivalence Class Testing

- Use ***Cartesian product*** of the normal variable ranges to identify test cases

Strong Normal Equivalence Class Testing

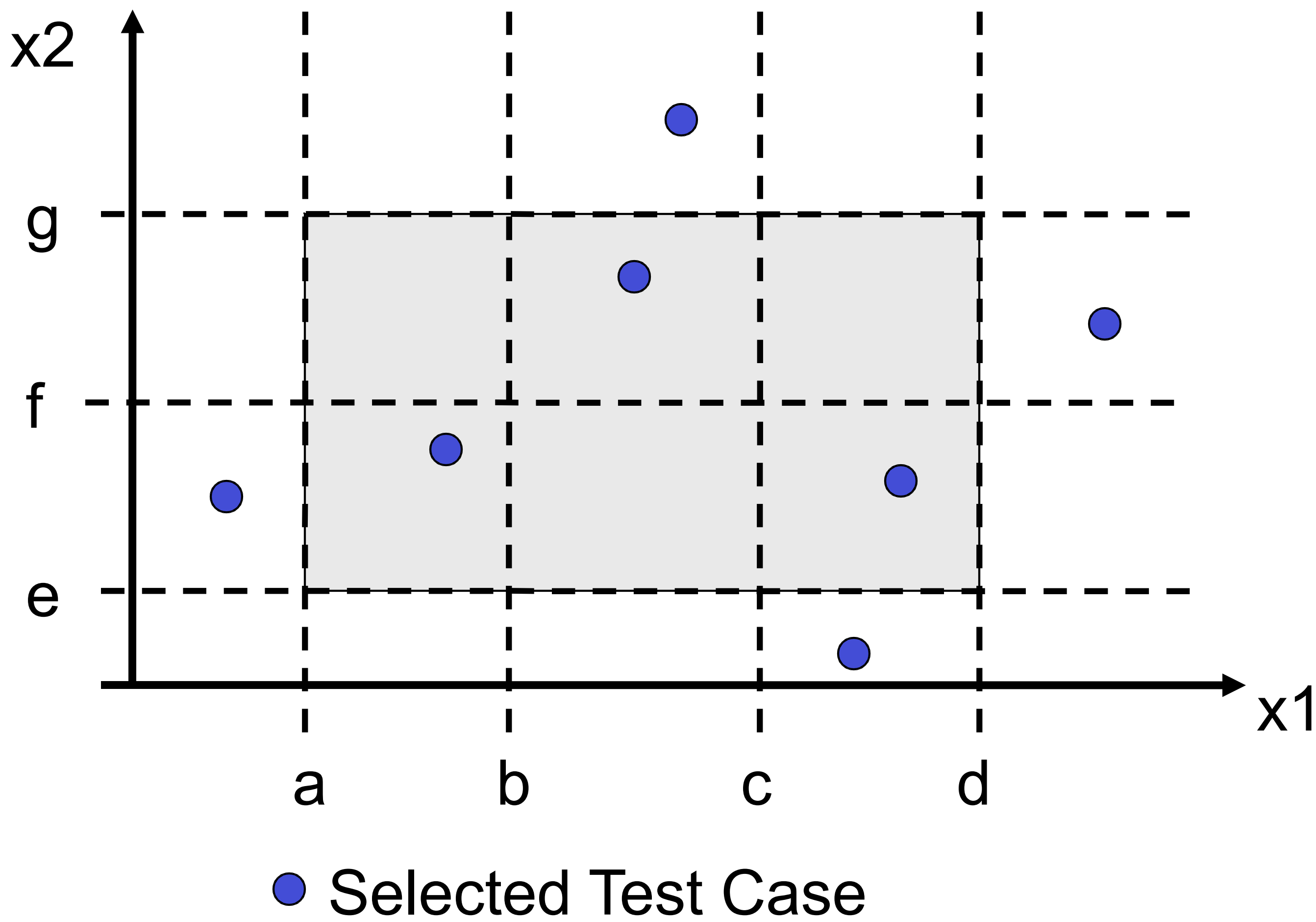


6 test cases

Weak Robust Equivalence Class Testing

- Valid Inputs
  - One value from each class
- Invalid Inputs
  - Each of these test cases will have one invalid value, the rest valid

Weak Robust Equivalence Class Testing

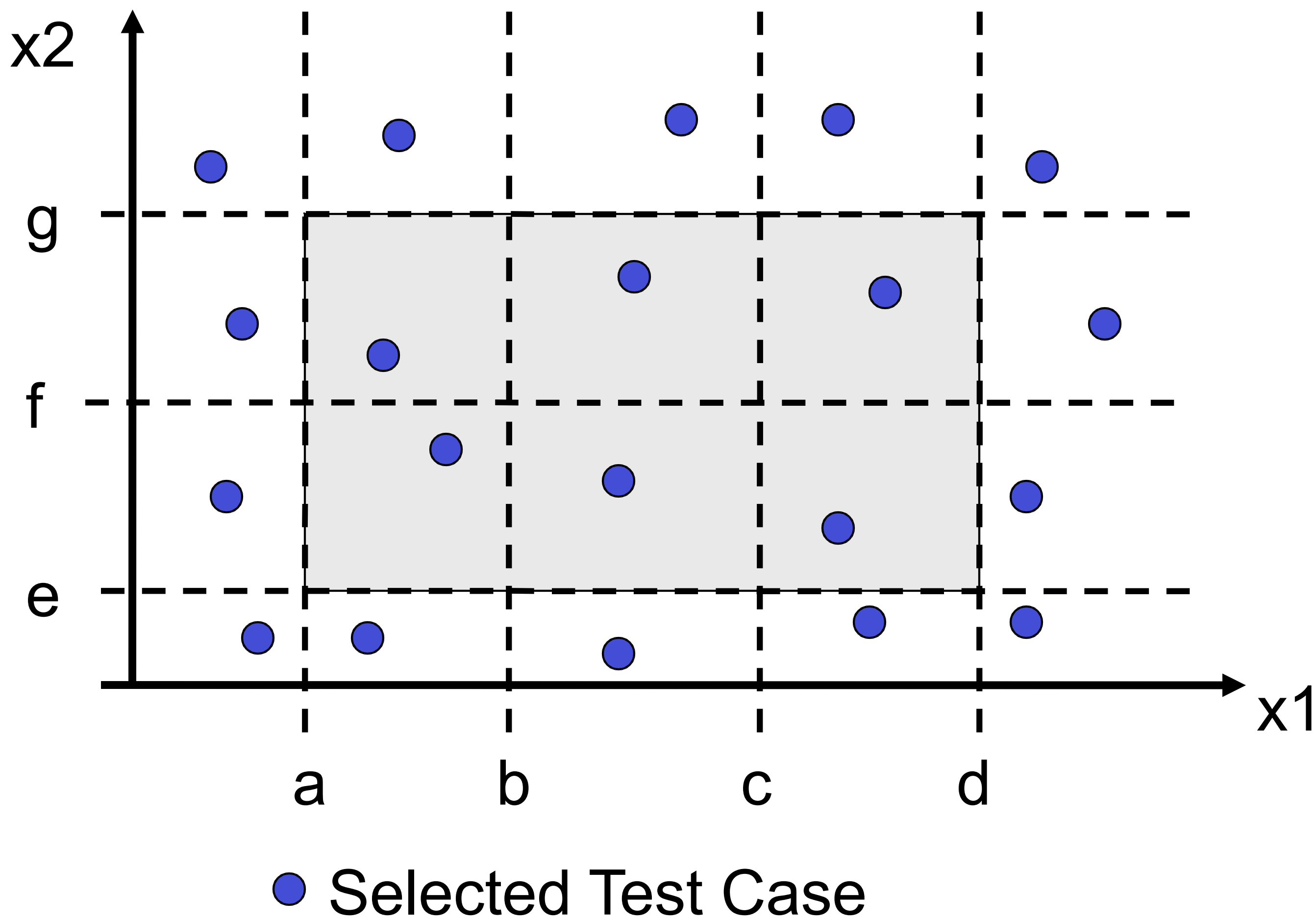


7 test cases

Strong Robust Equivalence Class Testing

- Cartesian product of all equivalence classes including valid and invalid data

Strong Robust Equivalence Class Testing



20 test cases

Equivalence Class Testing

- Can be applied when inputs or outputs are defined by intervals or sets of discrete values
- Equivalence criteria for partitioning determined by assuming a likely implementation
  - May require some repetition to identify a reasonable partitioning
- Often used in conjunction with BVA

Decision Tables*

- Identify conditions and actions
- Columns represent ***rules***
  - Columns correlate actions which should be taken for the corresponding conditions
- Derivation of test cases from decision tables
  - Interpret conditions as inputs
  - Interpret actions as outputs
  - ***Rules*** represent test cases

* *Software Testing: A Craftsman's Approach*, 2nd ed. by Paul C. Jorgensen

Decision Table Example - 1

Problem Description:*

A program accepts three integers ***a***, ***b***, and ***c*** input by the user and interprets these values as sides of a triangle. The program should determine what type of triangle is formed, if any:

Equilateral, Isosceles, Scalene, or NotATriangle

and output a corresponding message.

If ***a***, ***b***, and ***c*** form a triangle, they will satisfy the following relationships:

$a < b+c$, $b < a+c$, and $c < a+b$.

- If all three sides are equal, then an equilateral triangle is formed.
- If exactly one pair of sides is equal, then the triangle is isosceles.
- If no pair of sides is equal, the triangle is scalene.
- If any of the three inequalities does not hold, then the sides do not form a triangle.

*Example from *Software Testing: A Craftsman's Approach*, Paul Jorgensen

Decision Table Example - 2

Step 1: Identify conditions

- c1: $a < b+c$?
- c2: $b < a+c$?
- c3: $c < a+b$?
- c4: $a = b$?
- c5: $a = c$?
- c6: $b = c$?

Step 2: Identify actions

- a1: Not a triangle
- a2: Scalene
- a3: Isosceles
- a4: Equilateral
- a5: Impossible

Decision Table Example - 3

Step 3: Draw decision table

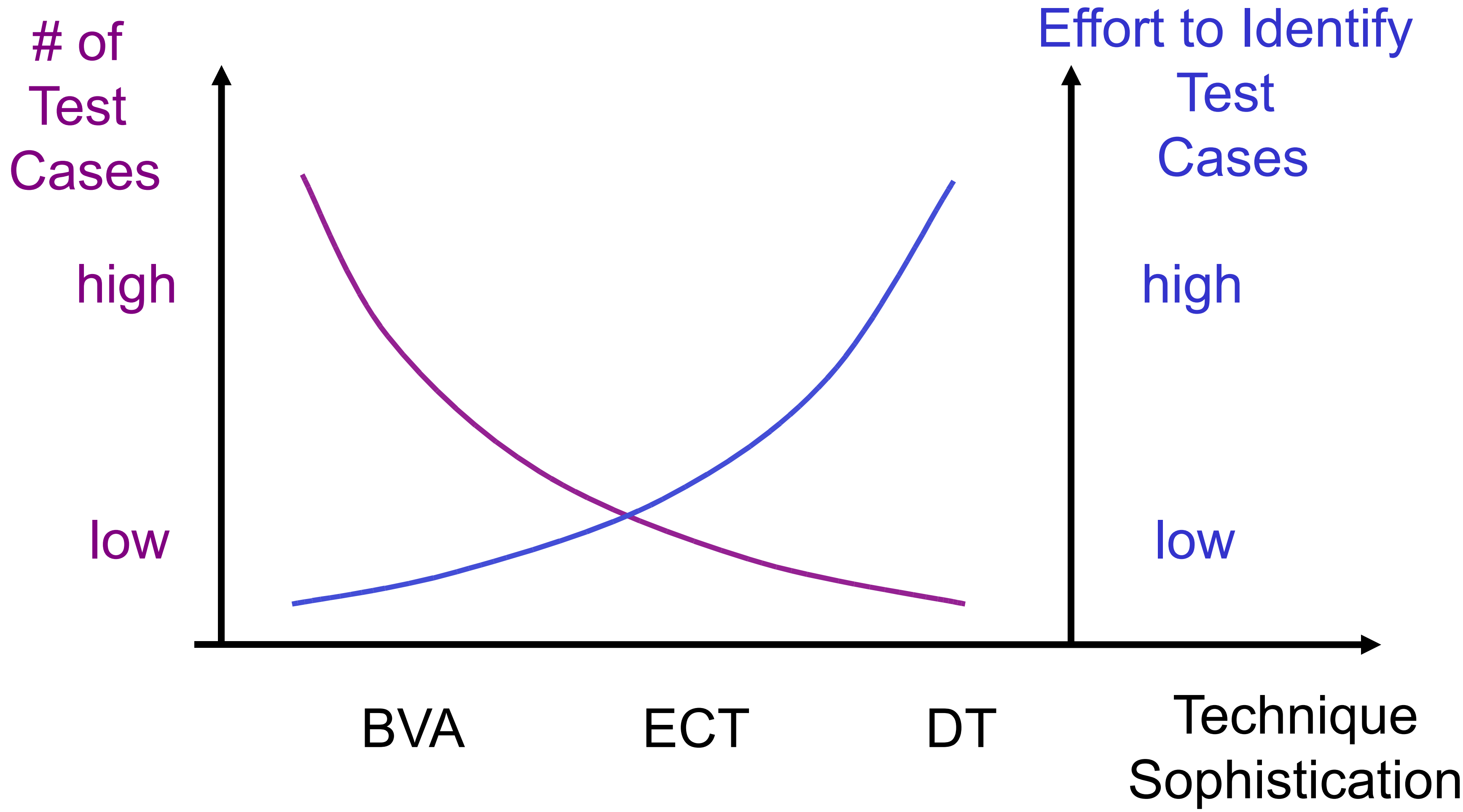
c1: a<b+c	F	T	T	T	T	T	T	T	T	T	T
c2: b<a+c	-	F	T	T	T	T	T	T	T	T	T
c3: c<a+b	-	-	F	T	T	T	T	T	T	T	T
c4: a=b	-	-	-	T	T	T	T	F	F	F	F
c5: a=c	-	-	-	T	T	F	F	T	T	F	F
c6: b=c	-	-	-	T	F	T	F	T	F	T	F
a1: Not a triangle	X	X	X								
a2: Scalene											X
a3: Isosceles							X		X	X	
a4: Equilateral				X							
a5: Impossible					X	X		X			

Decision Table Example - 4

Step 4: Identify Test Cases

Case ID	a	b	c	Expected Output
DT1	4	1	2	Not a Triangle
DT2	1	4	2	Not a Triangle
DT3	1	2	4	Not a Triangle
DT4	5	5	5	Equilateral
DT5	?	?	?	Impossible
DT6	?	?	?	Impossible
DT7	2	2	3	Isosceles
DT8	?	?	?	Impossible
DT9	2	3	2	Isosceles
DT10	3	2	2	Isosceles
DT11	3	4	5	Scalene

Functional Testing Techniques



Functional Testing Guidelines

c1: Variables (Physical or Logical)	P	P	P	P	P	L	L	L	L	L
c2: Independent Variables?	Y	Y	Y	Y	N	Y	Y	Y	Y	N
c3: Single Fault Assumption?	Y	Y	N	N	-	Y	Y	N	N	-
c4: Exception handling?	Y	N	Y	N	-	Y	N	Y	N	-
a1: Boundary Value Analysis		X								
a2: Robustness Testing	X									
a3: Worst-Case Testing				X						
a4: Robust Worst Case			X							
a5: Traditional Equivalence Class	X		X			X		X		
a6: Weak Equivalence Class	X	X				X	X			
a7: Strong Equivalence Class			X	X	X			X	X	X
a8: Decision Table					X					X

Table 8.1, *Software Testing: A Craftsman's Approach*, 2nd ed. by Paul C. Jorgensen

Structural Testing

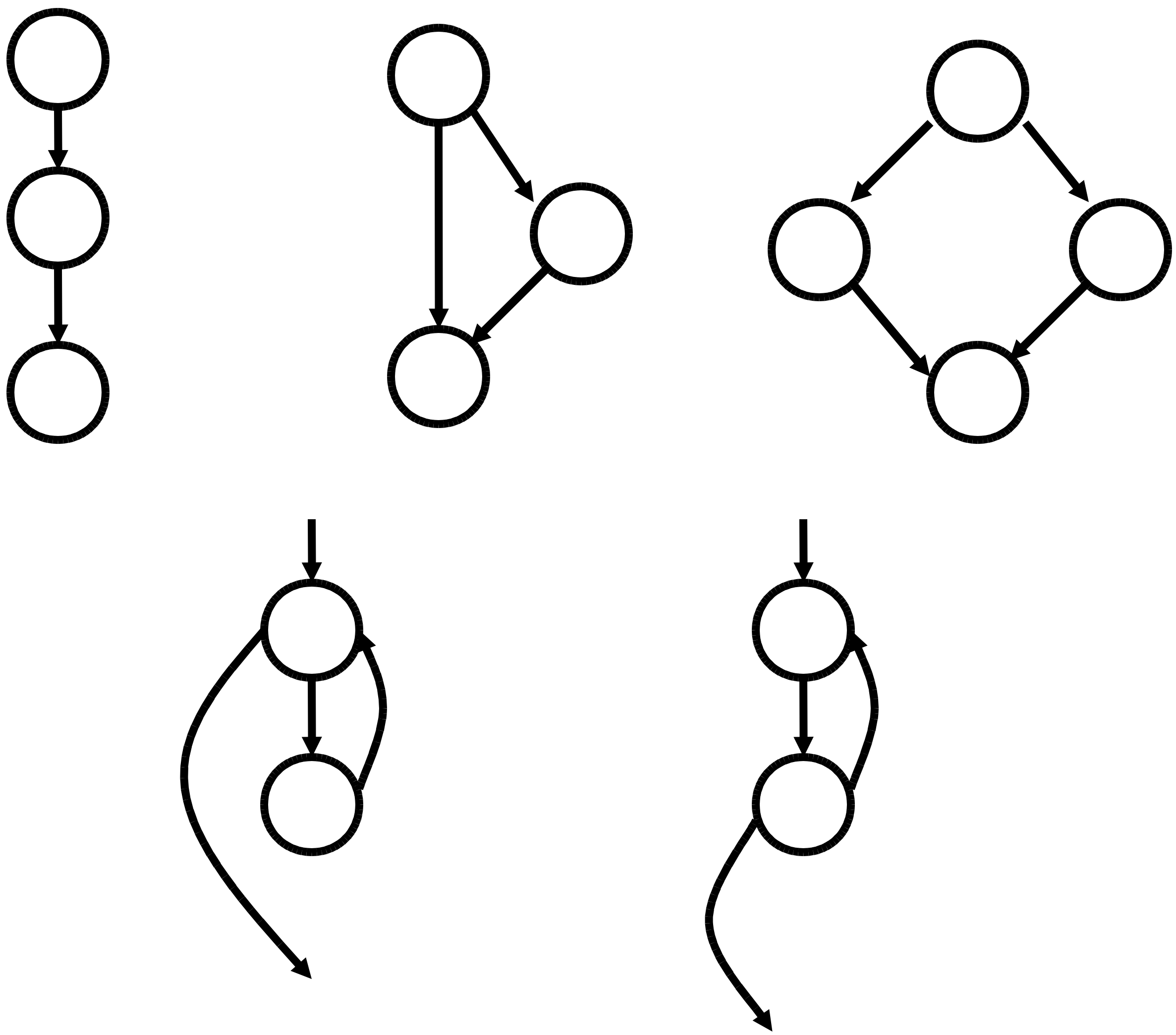
Why Structural Testing?

- Functional Testing Unknowns
  - ***Redundancy*** in the set of functional test cases
  - ***Gaps*** in the set of functional test cases

Structural Testing

- Test cases derived from the source code itself
- Metrics can quantify redundancy and gaps in the test set
- Structural testing is based upon graph theory
  - Source code statements may be viewed as a directed graph
  - Algorithms can be applied to determine number of possible paths, reachable/unreachable statements, etc.

Basic Control Structures as Graphs



Cyclomatic Complexity

- McCabe’s measure of product’s logical complexity
- It is computed as the number of binary decisions plus 1 (or E - N + 2)
- Cyclometric complexity is a good predictor of faults
- Some research shows that cyclomatic complexity has a high correlation with lines of code

Code Coverage

- Statement Coverage - Test cases should force each program statement to execute at least once (weakest form of code coverage)
- Condition Coverage - Test cases force each branch condition to evaluate to true and false at least once
- Multiple Condition Coverage - Test cases exercise both true and false outcomes of subexpressions within branch conditions
- Path Coverage - Test cases exercise all possible paths through the source code (strongest form of code coverage)