Chapter 20: Software Testing Black-Box Testing Techniques

Black-Box Testing

- It focuses on testing the functionality and behavior of the software.
- It considers classes of input that can effectively detect errors.
- It takes into account input values that often cause problems.
- It considers combination of input values that affects system operation.

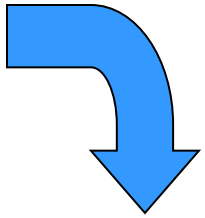
Black-Box Testing Techniques

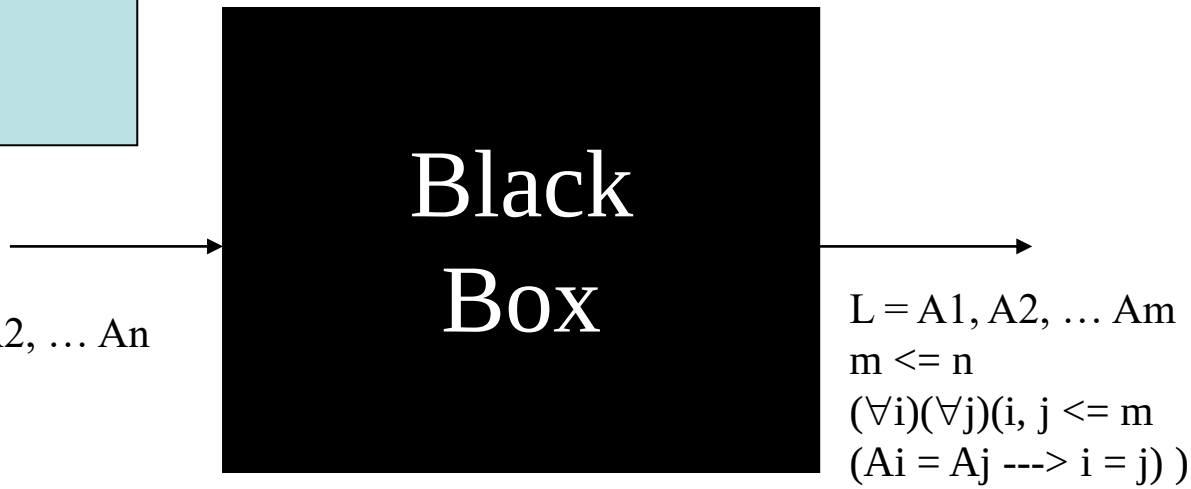
- Equivalence partitioning
- Boundary value analysis
- Cause-effect testing

Black-Box Testing Example

```
Procedure purge (var L:list)
        var p: …
(1)     begin p:= FIRST(L);
(2)     while P <> END(L) do
(3)     begin q:= next(p,L);
(4)        while q <> END(L) do
(5)           if Aq = Ap then
(6)              delete (Aq, L)
(7)           else q:= next(q,L);
(8)        end
(9)        p := next(p,L)
(10)    end;
```

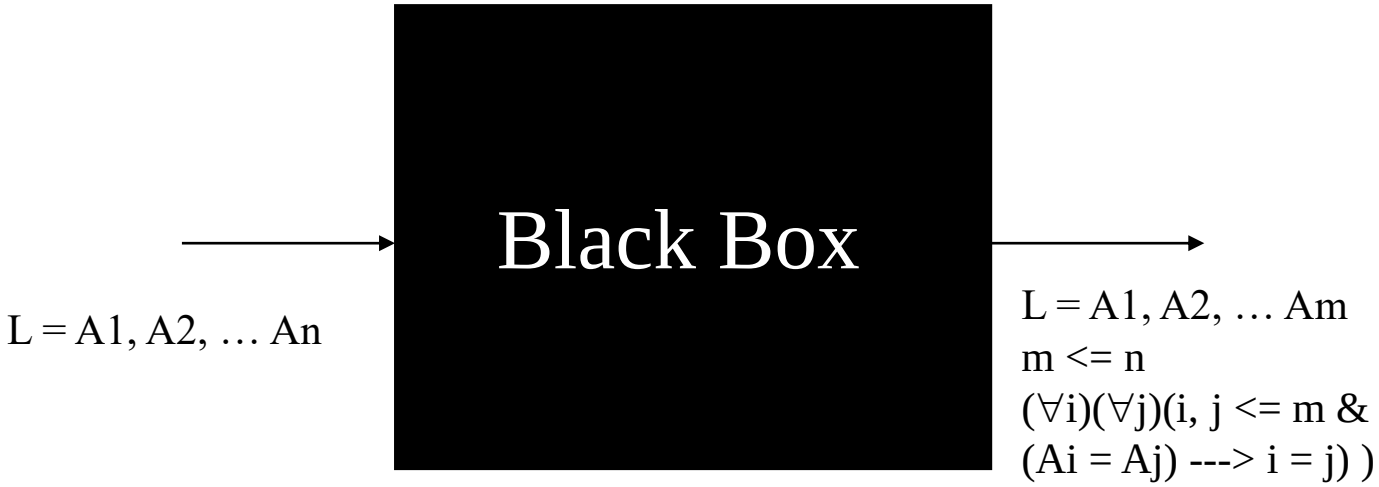
Black-box testing is based on the function performed.





Black-Box Testing – Analysis of Functionality

知道黑箱的功能，每個input,output的用途，但不知道內部如何做的，該怎樣做徹底的測試



L = A1, A2, … Am, m <= n

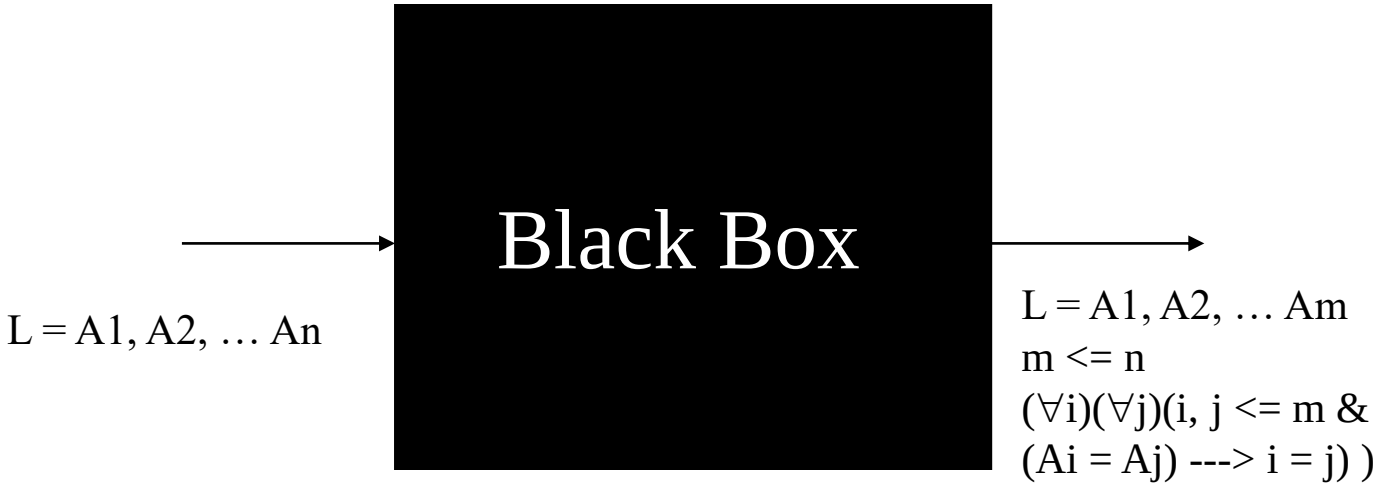
case 1: n=0. The input is an empty list. It implies m=n=0.

case 2: n=1. The input is a single element list. Expected: m=1. case 3: n>1.

3.1: m<n ⇒ L contains duplicate elements.

3.2: m=n ⇒ L does not contain duplicate elements.

Black-Box Testing – Deriving Test Cases



L = A1, A2, … Am, m <= n

case 1: n=0 (empty list). Input and expected output: L=( ).

case 2: n=1 (single-element list).

Input and expected output: L= (Ap).

case 3.1: m<n ⇒ L contains duplicate elements.

Input: L=(Ap, Aq, Ap), expected output: (Ap, Aq).

case 3.2: m=n ⇒ L not contain duplicate elements.

Input and expected output: L=(Ap Aq)

About the Example

- The post-condition is not adequate.
- It should also indicate that:
  - every unique element of the original list is also an element of the resulting list,
  - every element of the resulting list is also an element of the original list,
  - the elements of the resulting list are in the same order as they appear in the original list
- For simplicity, these are omitted.

Equivalence Partitioning

- Let S={a1, a2, ..., an} be a set of elements.
- $R \subseteq SXS$ is an equivalence relation, that is
  - reflexive: $\langle ai, ai \rangle \in R$
  - symmetric: $\langle ai, aj \rangle \in R \Rightarrow \langle aj, ai \rangle \in R$
  - transitive: $\langle ai, aj \rangle \in R \wedge \langle aj, ak \rangle \in R$

    $\Rightarrow \langle ai, ak \rangle \in R$
- Examples:
  1) S={0, 1, 2, ... }, $\langle x,y \rangle \in R$ if (x-y)%3=0. 被3除盡的數的集合是一個equivalence relation
  2) The relation of living in the same city.
  3) The relation of "having the same property"

Equivalence Partitioning

- Equivalence partitioning is a black box testing method.
- It divides the input domain of a program into disjoint subsets.
- Test cases are obtained by selecting an input value from each of the disjoint subsets.
- It is based on the belief that values from the same partition will detect similar errors – therefore, one test is enough for each partition.
- It reduces the total number of test cases.
- Equivalence partitioning is applicable for the output domain as well.

Equivalence Partitioning

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.

- Example:
  - input range is 0 -- 100, the equivalence classes are:
  - negative integers
  - integers between 0 and 100
  - integers greater than 100

Equivalence Partitioning

2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.

- Example:
  - Expected input condition is 10
  - valid partition is {10}
  - invalid partitions are
    - all integers that are less than 10
    - all integers that are greater than 10

Equivalence Partitioning

3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.

- Example:
  - expected input condition is “SunOS”
  - valid equivalence class is {“SunOS”}
  - invalid equivalence class is ALL_String - {“SunOS”}

4. If an input condition is boolean, one valid and one invalid class are defined.

Equivalence Partitioning

- Examples
  - Minimum GRE scores for admission
  - Phone numbers can be partitioned according to area code
  - Shipments can be partitioned according to service and zone
  - See page 506 for more examples
- Class Exercise:
  - Other examples of input conditions and equivalence classes?

Boundary Value Analysis

- Boundary value analysis is a test case generation technique that complements equivalence partitioning.
- Boundary value analysis leads to the selection of test cases at the “edges” of the partitions.
- Boundary value analysis derives test cases from the output domain as well.

Boundary Value Analysis

- if a range, (a,b), then values a,b, above a, above b, below a, below b all should be exercised.
- if a number of values, then min, max, above and below min, max all should be exercised.
- apply the two guidelines to output conditions.
- If internal program data structures have prescribed boundaries (e.g., an array has a defined limit of 100 entries), design a test case to exercise the data structure at its boundary.

Boundary Value Analysis

1. a < x < b;

   x = a + 1, x = b - 1, x = a, x = b

2. var A:array [1...100, 1…100] of integer;

   A[1,1] = 1 and others = 0

   do A[i,j] as 1 . Above

3. Special values:

   zero, 1, very large values.

   Values leading to output zeros.

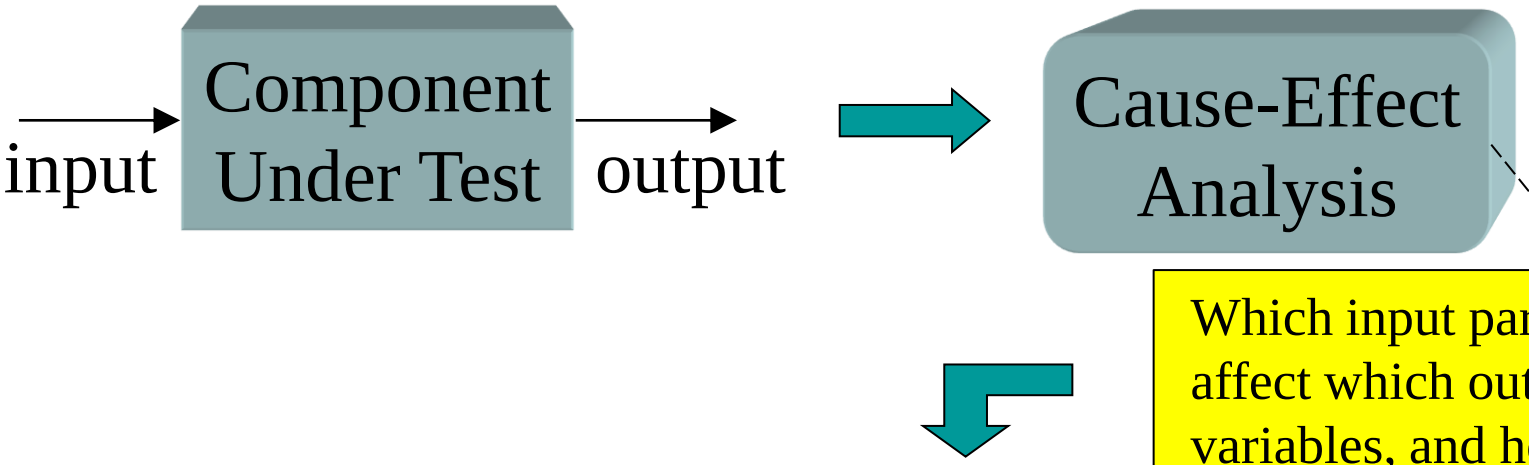
   Null array, null list/stack.

   Full array

Cause-Effect Testing

- It analyzes the functionality to identify the causes and their effects.
- A decision table is constructed to represent the cause-effect relationships.
- Rules of the decision table are used to derive test cases.

Cause-Effect Testing
一個人員去留鑑別程式



	1	2	3	4
10% Pay Raise	Y	N	N	N
Promotion	-	Y	Y	N
Own Office	-	Y	N	-
Case Count	4	1	1	2
Stay	x	x		
Quit			x	x

Four test cases can be derived.

Steps for Cause-Effect Testing

1. Causes (input conditions) and effects (actions) are listed for a module and an identifier is assigned to each.
2. A decision table (or decision tree) is generated that described which input combination leads to the execution of which action(s).
3. Decision table rules are converted to test cases. Test data are selected so that each rule in the table is exercised. Obviously, if a decision table has been used as a design tool, cause-effect analysis is no longer necessary.

Cause-Effect Analysis

- Checking completeness: The sum of the case count must equal to the number of all possible cases:

  sum of case count: 4+1+1+2=8

  number of all possible cases: $2^3=8$

	1	2	3	4
10% Pay Raise	Y	N	N	N
Promotion	-	Y	Y	N
Own Office	-	Y	N	-
Case Count	4	1	1	2
Stay	x	x		
Quit			x	x

Cause-Effect Analysis

- Checking consistency: no two rules can have the same condition combination but different action sequence.

rules

		1	2	3	4
condition stub	10% Pay Raise	Y	N	N	N
	Promotion	-	Y	Y	N
	Own Office	-	Y	N	-
	Case Count	4	1	1	2
action stub	Stay	x	x		
	Quit			x	x

Inspection and Testing

Inspection

- static
- a verification technique
- can check conformance to coding standards
- cannot check satisfiability of non-functional requirements

Testing

- dynamic
- a validation technique
- cannot check conformance to coding standards
- can check satisfiability of non-functional requirements