Testing

• Goals of Testing

• Classification
  – Test Coverage
  – Test Technique

• Blackbox vs Whitebox

• Real bugs and software bugs
Testing

• Primary objectives
  – Testing is a process of executing a software program with the intention of finding a error
  – A good test case is one that has a high probability of finding an as-yet undiscovered error
  – A successful test is one that uncovers an as-yet undiscovered error”
    (Glen Myers, "The art of software testing”)

• Secondary Objectives
  – Design tests that systematically uncover different classes of errors
  – Do so with a minimum of time and effort
  – Provide reliable indications of software quality

Test Techniques 1

• Classified according to the criterion used to measure the adequacy of a set of test cases:
  – Coverage-based testing
    • Testing requirements are specified in terms of the coverage of the product to be tested
  – Fault-based testing
    • Fault detecting ability of the test set determines the adequacy
  – Error-based testing
    • Focus on error-prone points, based on knowledge of the typical errors that people make
(Definitions)

- **Error**
  - Error is a human action that produces an incorrect result

- **Fault**
  - Consequence of an error is software containing a fault. A fault thus is the manifestation of an error

- **Failure**
  - If encountered, a fault may result in a failure

- What we observe **during testing** are failures

---

**PRS**

Exception handling is used to capture:

1. Errors
2. Faults
3. Failures
4. I am still sleeping ...
Test Techniques 2

• Or, classify test techniques based on the source of information used to derive test cases:

  – **White** (glass) **box** testing
    • Also called **structural** or program-based testing

  – **Black** **box** testing
    • Also called **functional** or specification-based testing

Black-Box Testing

• An approach to testing where the program is considered as a ‘black-box’

• The program test cases are based on the **system specification**

• Test planning can begin early in the software process
Black-Box Testing

Inputs causing anomalous behavior

System

Outputs which reveal the presence of defects

Input test data

Output test results

Equivalence Partitioning

- Input data and output results often fall into different classes where all members of a class are related

- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member

- Test cases should be chosen from each partition
Equivalence Partitioning

- Partition system inputs and outputs into ‘equivalence sets’
  - If input is a 5-digit integer between 10,000 and 99,999 equivalence partitions are
    - <10,000
    - 10,000-99,999
    - >99,999

- Choose test cases at the boundary of these sets
  - 00000, 09999, 10000, 99999, 100000
Equivalence Partitions

Input values

<table>
<thead>
<tr>
<th>Less than 4</th>
<th>Between 4 and 10</th>
<th>More than 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Less than 10000</th>
<th>Between 10000 and 99999</th>
<th>More than 99999</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999</td>
<td>10000</td>
<td>100000</td>
</tr>
</tbody>
</table>

Search Routine Specification

```plaintext
procedure Search (Key : Elem;
    T : Elem_Array;
    Found : in out Boolean;
    L : in out Elem_Index)

Pre-Condition
    -- the array has at least one element
    T'First <= T'Last

Post-Condition
    -- the element is found and is referenced by L
    ( Found and T(L) = Key)
    or
    -- the element is not in the array
    ( not Found and
    not (Exists I, T'First >= I <= T'Last, T (I) = Key ))
```
Testing Guidelines (Sequences)

- Test software with sequences which have only a single value
- Use sequences of different sizes in different tests
- Derive tests so that the first, middle and last elements of the sequence are accessed
- Test with sequences of zero length

Search Routine - Input Partitions

<table>
<thead>
<tr>
<th>Array</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single value</td>
<td>In sequence</td>
</tr>
<tr>
<td>Single value</td>
<td>Not in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>First element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Last element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Middle element in sequence</td>
</tr>
<tr>
<td>More than 1 value</td>
<td>Not in sequence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input sequence (T)</th>
<th>Key (Key)</th>
<th>Output (Found, L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
<td>true, 1</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>false, ??</td>
</tr>
<tr>
<td>17, 29, 21, 23</td>
<td>17</td>
<td>true, 1</td>
</tr>
<tr>
<td>41, 18, 9, 31, 30, 16, 45</td>
<td>45</td>
<td>true, 7</td>
</tr>
<tr>
<td>17, 18, 21, 23, 29, 41, 38</td>
<td>23</td>
<td>true, 4</td>
</tr>
<tr>
<td>21, 23, 29, 33, 38</td>
<td>25</td>
<td>false, ??</td>
</tr>
</tbody>
</table>

16
White Box Testing

- Also called Structural testing
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements

Objective is to exercise all program statements

White Box Testing

- Exercise all independent *paths* within a module at least once
- Exercise all logical *decisions* on their true and false sides
- Exercise all *loops* at their boundaries and within their operational bounds
- Exercise all internal data structures to assure their validity
Why White Box Testing

• Why not simply check that
  – Requirements are fulfilled?
  – Interfaces are available and working?

• Reasons for white-box testing:
  – logic errors and incorrect assumptions are inversely proportional to a path’s execution probability
  – we often believe that a path is not likely to be executed; in fact, reality is often counter intuitive
  – typographical errors are random; it’s likely that untested paths will contain some

Exhaustive Testing

There are $5^{20} = 10^{14}$ possible paths

If we execute one test per millisecond, it would take $3,170$ years to test this program

loop < 20x
Selective Testing

Basis Set

- **Basis set** of execution paths = set of paths that will execute all statements and all conditions in a program at least once

- **Cyclomatic complexity** defines the number of independent paths in the basis set

- Basis set is not unique

- **Goal**: Define test cases for basis set
Flow Graph Notation

Basis Path Testing

- Derive a logical complexity measure
  - Cyclomatic complexity $CV(G)$
    - Number of simple decisions +1 (compound decisions have to be split)
    - Number of enclosed areas +1 (uses flow-graph notation)
  - In this case, $CV(G) = 4$
- Use $CV(G)$ to define a basis set of execution paths
  - $CV(G)$ provides an lower bound of tests that must be executed to guarantee coverage of all programs
Cyclomatic Complexity

A number of industry studies have indicated that the higher CV(G), the higher the probability of errors.

Basis Path Testing

CV(G) = 4

There are four paths

- Path 1: 1,2,3,6,7,8
- Path 2: 1,2,3,5,7,8
- Path 3: 1,2,4,7,8
- Path 4: 1,2,4,7,2...7,8

We derive test cases to exercise these paths
Selective Testing

- Basis path testing
- **Condition testing**
- Loop testing
- Dataflow testing

Condition Testing

- Exercises each logical condition in a program module

- Possible conditions:
  - **Simple** condition:
    - Boolean variable (T or F)
    - Relational expression (a<b)
  
  - **Compound** condition:
    - Composed of several simple conditions ((a=b) and (c>d))
Condition Testing Methods

- **Branch** testing:
  - Each branch of each condition needs to be exercised at least once

- **Domain** testing:
  - Relational expression \( a<b \):
    - 3 tests: \( a<b, a=b, a>b \)
  - Boolean expression with \( n \) variables
    - \( 2^n \) tests required

Selective Testing

- Basis path testing
- Condition testing
- **Loop** testing
- Dataflow testing
Loop Testing

- Loops are the cornerstone of every program
- Loops can lead to non-terminating programs
- Loop testing focuses exclusively on the validity of loop constructs
  ```
  while X < 20 loop
    do something
  end loop;
  ```
Testing Simple Loops

- Minimum conditions - simple loops
  - skip the loop entirely
  - only one pass through the loop
  - two passes through the loop
  - m passes through the loop \( m < n \)
  - \((n-1), n, \) and \((n+1)\) passes through the loop

\( n = \) maximum number of allowable passes

Testing Nested Loops

- Just extending simple loop testing
  - number of tests grows geometrically

- Reduce the number of tests:
  - start at the innermost loop; set all other loops to minimum values
  - conduct simple loop test; add out-of-range or excluded values
  - work outwards while keeping inner nested loops to typical values
  - continue until all loops have been tested
Testing Concatenated Loops

- Loops are independent of each other:
  - Use simple-loop approach
- Loops depend on each other:
  - Use nested-loop approach

Testing Unstructured Loops

Bad Programming!
Selective Testing

- Basis path testing
- Condition testing
- Loop testing
- Dataflow testing

Dataflow Testing

- Partition the program into pieces of code with a single entry/exit point
- For each piece find which variables are set/used
- Various covering criteria:
  - For all set-use pairs
  - For all set to some use