CS 4387/5387
SOFTWARE V&V

LECTURE 4
BLACK-BOX TESTING

Outline

- Quiz
- Black-Box Testing
  - Equivalence Class Testing (Equivalence Partitioning)
  - Boundary value analysis
  - Decision Table Testing
Quiz - 1

1. read (x)
2. read (z)
3. if x != 0 then {
   4.   y = x * z;
   5.   x = z
}
6. else y = z
7. if y > 1 then
8.   print (y * z)
9. else print (y * x)

Cyclomatic complexity:
• Upper limit on the number of test cases required to ensure branch coverage
• Cyclomatic complexity= E - N + 2 = 12 - 11 + 2 = 3
• B + 1 = 2 + 1 = 3

Quiz - 2

1. read (x)
2. read (z)
3. if x != 0 then {
   4.   y = x * z;
   5.   x = z
}
6. else y = z
7. if y > 1 then
8.   print (y * z)
9. else print (y * x)

Cyclomatic complexity:
• Upper limit on the number of test cases required to ensure branch coverage
Quiz – 3

Def-use coverage: every path from every definition of every variable to every use of that definition is exercised in some test.

1. read (x)
2. read (z)
3. if x != 0 then {
4. y = x * z;
5. x = z }
6. else y = z
7. if y > 1 then
8. print (y * z)
9. else print (y * x)

Def: x
Use: x

Def: y
Use: x, z

Def: x
Use: z

Def: x
Use: x

Quiz - 4

Path Coverage: Every distinct path through code is executed at least once

1. read (x)
2. read (z)
3. if x != 0 then {
4. y = x * z;
5. x = z }
6. else y = z
7. if y > 1 then
8. print (y * z)
9. else print (y * x)

Test Paths:
1, 2, 3, 4, 5, J, 7, 8, J.
1, 2, 3, 4, 5, 7, 9, J.
1, 2, 3, 6, 7, 9, J.
1, 2, 3, 6, 7, 7, 9, J.

Test Case (x, z): (1, 2)
Test Case (x, z): (1, 1)
Test Case (x, z): (0, 2)
Test Case (x, z): (0, 1)
Unit & Integration Testing

- Objective: to ensure that code implemented the design properly.

Unit Testing

- Code Inspections
- Code Walkthroughs
- Black-box Testing
- White-box Testing
Unit and Integration Testing

Test Oracles

- Two types of oracles
  - Human: an expert that can examine an input and its associated output and determine whether the program delivered the correct output for this particular input.
  - Automated: a system capable of performing the above task.
Testing can be viewed as selecting different colored balls from an urn where:
- Black ball = input on which program fails.
- White ball = input on which program succeeds.
- Only when testing is exhaustive is there an “empty” urn.

A program that always fails

A correct program

A typical program
Black-box/Closed-box Testing

- Incorrect or missing functions
- Interface errors
- Performance error

Black-Box Testing Techniques

- Definition: a strategy in which testing is based on requirements and specifications.

- Applicability: all levels of system development
  - Unit
  - Integration
  - System
  - Acceptance

- Disadvantages: never be sure of how much of the system under test (SUT) has been tested.

- Advantages: directs tester to choose subsets to tests that are both efficient and effective in finding defects early in development.
Black-Box Testing

- Exhaustive testing
- Equivalence class testing
- Boundary value analysis
- Decision table testing
- Pairwise testing
- State-Transition testing
- Domain analysis testing
- Use case testing

Exhaustive Testing

- Definition: testing with every member of the input value space.
- Input value space: the set of all possible input values to the program.
- How Feasible?

```c
int increment (int a) {
    ............
}
```

If one test takes 1 ms how much time it will take to test exhaustively function above?

No of possible inputs: From −2,147,483,648 to 2,147,483,647, from −(2^{31}) to 2^{31} − 1 = 2^{32} − 1 possible inputs

Time to test

```c
long increment (long a) {
    ............
}
```

Time to test

- 2^{32} − 1 ms = (2^{32} − 1)/1000 sec = 1.6 months
- 2^{64} − 1 ms = (2^{64} − 1)/1000 sec = 584,542,046.1 years
Equivalence Class Testing

- Testing technique used to reduce the number of test cases to a manageable level while still maintaining reasonable test coverage.

- Each EC consists of a set of data that is treated the same by the module or that should produce the same result.

- Any data value within a class is equivalent, in terms of testing, to any other value.

---

Equivalence Class Testing Technique

Partition test cases into classes such that:

1. Every possible input belongs to one of the classes
2. No input belongs to two different classes
3. If we demonstrate a fault in the code for a given input, we should demonstrate a fault with any other input from the same class (with a high probability)
4. Classes are identified by looking at boundary values for the variables of the application
Example-1

- If \( x > y \) then \( S_1 \) else \( S_2 \)

Equivalence classes for values of \( x \) and \( y \):

- \( x > y \)  
- \( x \leq y \)

Example - 2

- Taking each input condition (usually a sentence or phrase in the specification) and partitioning it into two or more groups:

- Input condition
  - range of values \( x \): 1-50

- Valid equivalence class
  - \(? < x < ?\)

- Invalid equivalence classes
  - \( x < ? \)
  - \( x > ? \)
Example - 2

- Taking each input condition (usually a sentence or phrase in the specification) and partitioning it into two or more groups:

- Input condition
  - range of values x: 1-50

- Valid equivalence class
  - $0 < x < 51$

- Invalid equivalence classes
  - $x < 1$
  - $x > 50$

Example 3 (Group Exercise)

- The program specification states that the program accepts 4 to 10 inputs which are five-digit integers greater than 10,000.

What are the partitions?
Example 3-Partitions

The program specification states that the program accepts 4 to 10 inputs which are five-digit integers greater than 10,000.

\[
\begin{array}{ccc}
X < 4 & 4 \leq X \leq 10 & X > 10 \\
\end{array}
\]

\[X = \text{Number of input values}\]

\[
\begin{array}{ccc}
Y < 10,000 & 10,000 \leq Y \leq 99,999 & Y > 99,999 \\
\end{array}
\]

\[Y = \text{Actual input value}\]

Discussion Point

- Consider a module for human resources system that decides how to handle employment process based on the applicant’s age. The rules are:
  - 0-16: Don’t Hire
  - 16-18: Can hire only as part-time
  - 18-55: Can hire as a full-time
  - 55-99: Don’t hire
- Typical test would be (15, Don’t Hire)
- How about test cases like (969, …), (FRED, …), (&#$@, …)?
### Design-by-Contract vs. Defensive Design - 1

- Testing based on design-by-contract relies on the module’s pre and post conditions.
  - **Pre condition:** defines what the module requires for it to accomplish its task.
    - Example: for a module that reads a file, an appropriate pre condition would be that the file is readable.
  - **Post condition:** defines the conditions that must be met if the module completes its task(s) successfully.
    - Example: file is open and ready for operations.
  - In testing based on design-by-contract, you have to worry only about testing the conditions satisfied by the pre condition.

### Design-by-Contract vs. Defensive Design - 2

- In defensive design, the module is designed to accept any input.
  - If pre conditions are met, the module will achieve its post conditions.
  - Otherwise, the module will notify the caller by returning an error code or throwing an exception.

- As a tester, you have to consult with designers in order to understand which approach is being used.
  - Even though they might not be even aware of it.
Equivalence Class Testing: Guidelines

1. If an input condition specifies a range of values; identify one valid EC and two invalid EC.
2. If an input condition specifies the number (e.g., one through 6 owners can be listed for an automobile); identify one valid EC and two invalid EC (- no owners; - more than 6 owners).
3. If an input condition specifies a set of input values and there is reason to believe that each is handled differently by the program; identify a valid EC for each set and one invalid EC for the other sets.
4. If an input condition specifies a “must be” situation (e.g., first character of the identifier must be a letter); identify one valid EC (it is a letter) and one invalid EC (it is not a letter)
5. If there is any reason to believe that elements in an EC are not handled in an identical manner by the program, split the equivalence class into smaller equivalence classes.

Identifying Test Cases

- Assign a unique number to each EC.

- Until all valid ECs have been covered by test cases, write a new test case covering as many of the uncovered valid ECs as possible.

- Until all invalid ECs have been covered by test cases, write a test case that cover one, and only one, of the uncovered invalid ECs.
Group Exercise

Problem statements: NextDate is a function of three variables: month, day and year. It returns the date of the day after the input date. The month, day and year variables have integer values subject to these conditions:

\[ \begin{align*}
C1: & \quad 1 \leq \text{month} \leq 12 \\
C2: & \quad 1 \leq \text{day} \leq 31 \\
C3: & \quad 1812 \leq \text{year} \leq 2012
\end{align*} \]

Valid ECs:

Invalid Ecs:

The craft of testing is to combine test cases to test as many ECs as possible within a single test.

But this might cause a problem. Any ideas why?

---

Group Exercise - Answers

Valid ECs:

\[ \begin{align*}
\text{M1:} & \quad \{\text{Month: } 1 \leq \text{Month} \leq 12\} \\
\text{D1:} & \quad \{\text{Day: } 1 \leq \text{Day} \leq 31\} \\
\text{Y1:} & \quad \{\text{Year: } 1812 \leq \text{Year} \leq 2012\}
\end{align*} \]

Invalid Ecs:

\[ \begin{align*}
\text{M2:} & \quad \{\text{Month: } \text{Month} > 1\} \\
\text{M3:} & \quad \{\text{Month: } \text{Month} > 12\} \\
\text{D2:} & \quad \{\text{Day: } \text{Day} > 31\} \\
\text{Y2:} & \quad \{\text{Year: } \text{Year} > 1812\} \\
\text{Y3:} & \quad \{\text{Year: } \text{Year} > 2012\}
\end{align*} \]

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
<th>Ecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/15/2010</td>
<td>10/16/2010</td>
<td>M1, D1, Y1</td>
</tr>
<tr>
<td>0/15/2010</td>
<td>Invalid</td>
<td>M2</td>
</tr>
<tr>
<td>14/15/2010</td>
<td>Invalid</td>
<td>M3</td>
</tr>
<tr>
<td>10/0/2010</td>
<td>Invalid</td>
<td>D2</td>
</tr>
<tr>
<td>10/35/2010</td>
<td>Invalid</td>
<td>D3</td>
</tr>
<tr>
<td>10/10/1800</td>
<td>Invalid</td>
<td>Y2</td>
</tr>
<tr>
<td>10/10/2020</td>
<td>Invalid</td>
<td>Y3</td>
</tr>
</tbody>
</table>
Consider the example of the Goofy Mortgage Company (GMC).

**Mortgages are allowed for:**
- A person with income between $1,000 and $83,333
- Condominiums, Townhouses, Single family dwellings
- 1 to 5 dwellings in a single mortgage

**Mortgages are not allowed for:**
- Corporations, Trusts, or Partnerships
- Duplex, Mobile Home, or Tree houses
- Less than one dwelling or more than 5

<table>
<thead>
<tr>
<th>Monthly Income</th>
<th># Dwellings</th>
<th>Applicant</th>
<th>Dwelling Type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>2</td>
<td>person</td>
<td>townhouse</td>
<td>Grant Mortgage</td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>person</td>
<td>townhouse</td>
<td>Deny Mortgage</td>
</tr>
<tr>
<td>2500</td>
<td>7</td>
<td>person</td>
<td>townhouse</td>
<td>Deny Mortgage</td>
</tr>
<tr>
<td>2500</td>
<td>2</td>
<td>Corporation</td>
<td>townhouse</td>
<td>Deny Mortgage</td>
</tr>
<tr>
<td>2500</td>
<td>2</td>
<td>Person</td>
<td>Mobile Home</td>
<td>Deny Mortgage</td>
</tr>
</tbody>
</table>
Applicability and Limitations

- Most suited to systems in which much of the input data takes on values within ranges or within sets.

- It makes the assumption that data in the same EC is, in fact, processed in the same way by the system.

- EC testing is equally applicable at the unit, integration, system, and acceptance test levels. All it requires are inputs or outputs that can be partitioned based on the system’s requirements.

Boundary Value Testing

- Equivalence class testing has the advantage of reducing the number of test case

- But it also has the other advantage of alerting us to the notion of boundary testing

- Boundary value testing focuses on the boundaries simply because that is where so many defects hide. Defects can be in the requirements, design, or code.

- The most efficient way of finding such defects, either in the requirements or the code, is through inspection (Software Inspection, Gilb and Graham’s book).

Going back to the HR system example:

- 0-16: Don’t Hire
- 16-18: Can hire only as part-time
- 18-55: Can hire as a full-time
- 55-99: Don’t hire
Boundary Value Testing - Technique

- Identify the ECs.

- Identify the boundaries of each EC.

- Create test cases for each boundary value by choosing:
  - one point on the boundary,
  - one point just below the boundary,
  - and one point just above the boundary.

Boundary Value Testing - Example

<table>
<thead>
<tr>
<th>Less than 10000</th>
<th>Between 10000 and 99999</th>
<th>More than 99999</th>
</tr>
</thead>
</table>

What are the boundary values to test?
Boundary Value Testing - Example

What are the boundary values to test?

Boundary Value Analysis Test Cases for Triangle program (1 <= a, b, c <= 200)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>Invalid input</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>1</td>
<td>Isosceles</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>100</td>
<td>2</td>
<td>Isosceles</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
<td>199</td>
<td>Isosceles</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>200</td>
<td>Isosceles</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>100</td>
<td>201</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
Group Exercise

Given the following method specification:

```java
/** Returns true iff key is contained in values. */
//@ requires values != null;
//@ ensures \result == (\exist int i; i >= 0 && i < values.length; 
//@ values[i] == key);
public static boolean contains(int[] values, int key)
```

Perform boundary analysis to select a reasonable set of test cases.

---

Answer

Test suite:
- `{}`, 2>
- `{2,3,24,5,6,6,7,22,6}, 9>
- `{2,3,24,5,6,6,7,22,6}, 2>
- `{2,3,24,5,6,6,7,22,6}, 33>
- `{2,3,24,5,6,6,7,22,6}, 22>
- `{2,3,24,5,6,6,7,22,6}, 6>`
Applicability and Limitations

- Boundary value testing is equally applicable at the unit, integration, system, and acceptance test levels.
- All it requires are inputs that can be partitioned and boundaries that can be identified based on the system’s requirements.

Decision Table Testing

- Decision tables are an excellent tool to capture certain kinds of system requirements and to document internal system design.
- They are used to record complex business rules that a system must implement.
- In addition, they can serve as a guide to creating test cases.
Technique: General Format of a Decision Table

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>...</th>
<th>Rule P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition-m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action-n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Don’t Care Entries

<table>
<thead>
<tr>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rules 3,4</th>
<th>Rule 5</th>
<th>Rule 6</th>
<th>Rules 7,8</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>C3</td>
<td>T</td>
<td>F</td>
<td>_</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>A1</td>
<td>X</td>
<td>X</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>A2</td>
<td>X</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>A3</td>
<td>X</td>
<td>X</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>A4</td>
<td>X</td>
<td>X</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

The don’t care entry has two interpretations:
- the condition is irrelevant, or
- the condition does not apply. Sometimes the “n/a” symbol for this latter interpretation.
Types of Decision Tables

- Limited entry decision tables:
  - all the conditions are binary

- Extended entry decision tables:
  - conditions are allowed to have several values.

- Decision tables are deliberately declarative (as opposed to imperative);
  - no particular order is implied by the conditions, and selected actions do not occur in any particular order.

Decision Table-Based Testing

- Applicable for requirements in if-then
  - if $C_1$ and $C_2$ and...and $C_n$ then $A_k$

- Create a decision table comprised of conditions and actions.

- Number of columns: $2^n$ (n is number of conditions)

- Number of rows: $n + m$, where m is number of actions

- For each set of conditions, there is a corresponding action
Group Exercise: Triangle program

The program accepts three integers, \(a\), \(b\), and \(c\) as input. The three values are interpreted as representing the lengths of sides of a triangle. The integers \(a\), \(b\), and \(c\) must satisfy the following conditions:

- \(a < b + c\)
- \(b < a + c\)
- \(c < a + b\)
- \(1 \leq a \leq 200\)
- \(1 \leq b \leq 200\)
- \(1 \leq b \leq 200\)

- If the integers \(a\), \(b\), and \(c\) do not constitute a triangle, we do not even care about possible equalities (rule 1).
- If two pairs of integers are equal, by transitivity, the third pair must be equal; thus the negative entry (N) makes these rules impossible (rule 3).

---

Decision Table to Test Cases

<table>
<thead>
<tr>
<th>Input</th>
<th>Test Case 1</th>
<th>Test Case 2</th>
<th>...</th>
<th>Test Case 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expected Results</td>
<td>Action-1</td>
<td>Action-2</td>
<td>...</td>
<td>Action-9</td>
</tr>
</tbody>
</table>

- If \(a\), \(b\), and \(c\) form a triangle? \(N\) \(Y\) \(Y\)
- \(a + b \leq d\) \(-\) \(Y\) \(Y\)
- \(a + c \leq d\) \(-\) \(Y\) \(Y\)
- \(b + c \leq d\) \(-\) \(Y\) \(N\)

- A1: Not a triangle \(X\)
- A2: Scalene
- A3: Isosceles
- A4: Equilateral \(X\)
- A5: Impossible \(X\)
Develop Decision Table for the Triangle Problem

What are the conditions?
- C1: a, b, c form a triangle?
- C2: a = b?
- C3: a = c?
- C4: b = c?

What are the actions?
- Not a triangle, scalene, isosceles, equilateral, impossible

How many rows?
- # of rows = #conditions + #actions

How many columns?
- 2 # of conditions
- Or is it?

<table>
<thead>
<tr>
<th>C1: a, b, c form a triangle?</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>C2: a = b?</td>
<td>DC</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>C3: a = c?</td>
<td>DC</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>C4: b = c?</td>
<td>DC</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

| A1: Not a Triangle          | X  |     |    |    |    |    |    |    |    |
| A2: Scalene                 |    |     |    |    |    |    |    |    |    |
| A3: Isosceles               |    | X   | X  | X  |    |    |    |    |    |
| A4: Equilateral             |    |    |    |    |    |    |    |    |    |
| A5: Impossible              | X  | X   | X  |    |    |    |    |    |    |
Next Week

- Quiz
- More on Decision Tables
- Pair Wise Testing
- Use-Case Testing