White Box-based Coverage Testing

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong
**Speaker Biographical Sketch**

- Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas

- Vice President, IEEE Reliability Society

- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)

- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project: *Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses*

- Founder & Steering Committee co-Chair for the SERE conference *(IEEE International Conference on Software Security and Reliability)*
  (http://paris.utdallas.edu/sere12)
Two Techniques for Test Generation

- Statement coverage
- Decision coverage
Preliminary
Example I

• How to search for stones inside a rectangular field?
  – We divide the entire field into *smaller cells*
Quiz

• How should the field be divided?
• How should we organize the search?
Analogy

- Field → Program
- Stone → Program bug
- Scan a cell → Test program on one input
- Find a stone → Find a program bug
- Partition → Subset of input domain
- Rectangle size → Size of a partition (with respect to “Program”)
Coverage Principle

- The basic idea of coverage testing is that testing is complete when a well-defined set of tests is complete.
  - Example
    - Pilots use pre-flight check lists
    - Shoppers use grocery lists
    - To assure the correct completion of their tasks
  - In the same way testers can count the completed elements of a test plan
    - Examples
      - Requirements
      - Statements
      - Decisions
The Role of Coverage in Testing

- It provides a way of monitoring and measuring the progress of testing against explicit quantitative completion criteria
  - Gives a clear measure of the completion of the testing task
  - Example, for a computer program
    - Requirements
    - Statements
    - Decisions
What is Code Coverage Testing

• It is “White Box Testing”

• Takes into account the structure of the software being tested

• Measures how thoroughly the code has been tested with respect to certain metrics
When test inputs are generated using program specifications, we say that we are doing functional testing.

- Functional testing tests how well a program meets the functionality requirements.

These two types of testing are complementary.

- Basic functionalities should always be tested.
- The set of tests generated from function testing provides a good basis for code coverage testing.
**Importance of Code Coverage Testing**

- In general, a piece of code must be executed before a fault in it can be exposed.

- *Helps early fault detection*
  - Are system testers finding faults that should have been found and fixed by developers?
  - Relative cost of fixing a software fault
Statement Coverage
Statement Coverage: Computation

- The statement coverage of a test set $T$ with respect to $(P, R)$ is computed as $S_c/(S_e-S_i)$, where
  - $S_c$ is the number of statements covered,
  - $S_i$ is the number of infeasible (unreachable) statements, and
  - $S_e$ is the total number of executable statements in the program to be tested.

- $T$ is considered adequate with respect to the statement coverage criterion if the statement coverage of $T$ with respect to $(P, R)$ is 1 (i.e., 100%).
Statement Coverage: An Example

begin
    int X, Y;
    int Z;
    input (X, Y);
    Z = 0;
    if (X < 0 and Y < 0) {
        Z = X*Y;
        if (Y >= 0)
            Z = Z + 1;
    }
    else
        Z = X*X*X;
    output (Z);
end
Statement Coverage: An Example (cont’d)

- $S_e = \{4, 5, 6, 7, 8, 9, 12, 13\}$
  Let $T_1 = \{t_1: <x = -1, y = -1>, t_2: <x = 1, y = 1>\}$

- Statements covered:
  - $t_1$: follow the red line
  - $t_2$: follow the green line

- $S_c = 7$, $S_i = 1$, $S_e = 8$. The statement coverage for $T_1$ is $7/(8 - 1) = 1$

- Note that 9 is infeasible

- Thus, we conclude that $T_1$ is adequate with respect to the statement coverage criterion.
Decision Coverage
**Decision versus Simple Condition**

- A decision can be composed of *a simple condition* such as $x<0$, or of *a more complex condition*, such as $(x<0 \text{ AND } y<0)$.

- A simple condition is considered covered if it evaluates to *true* and *false* in one or more executions of the program in which it occurs.

- A *compound condition* is considered covered if each *simple condition* it is comprised of is also covered.

```
1  if (x < 0 and y < 0) {
2    z=foo(x,y);
}

1  if (x<0)
2    if (y<0)
3    z=foo(x,y);
```

one decision & two simple conditions

two decisions & two simple conditions
How to Cover a Decision?

- A decision is considered **covered** if the flow of control has been diverted to **all possible destinations** that correspond to this decision, i.e., **all outcomes of the decision have been taken**.

- This implies that, for example, the expression in the *if* or a *while* statement has evaluated to *true* in some execution of the program under test and to *false* in the same or another execution.

  - Examples
    - if (x < 0) { ..... }
    - if ((x<0) and (y>0) { ..... }
    - while (x < 3) { ..... }
**Decision Coverage: Computation**

- The decision coverage of a test set $T$ with respect to $(P, R)$ is computed as $D_c/(D_e-D_i)$, where
  - $D_c$ is the number of decisions covered,
  - $D_i$ is the number of *infeasible decisions*, and
  - $D_e$ is the *total number of decisions* in the program to be tested.

- $T$ is considered adequate with respect to the decision coverage criterion if the decision coverage of $T$ with respect to $(P, R)$ is 1 (i.e., 100%).
Quiz

• Consider the following program and a test set

```plaintext
1 begin
2   int x, y, z;
3   input (x, y);
4   if(x<0 or y<0) T = \{ t_1: < x = -3 y = -2 > \}
5       z=foo-1(x,y);
6   else
7       z=foo-2(x,y);
8   output(z);
9 end
```

• Is $T$ adequate with respect to decision coverage?
Statement Coverage versus DecisionCoverage
A Program with a Bug

- This following program inputs an integer \( x \)
  - if \( x < 0 \), transforms it into a positive value before invoking \( \text{foo-1} \) to compute the output \( z \)
  - if \( x \geq 0 \), compute \( z \) using \( \text{foo-2} \)

```
begin
  int x, z;
  input (x);
  if(x<0)
    x = -x;
  z=foo-1(x);
  output(z);
end
```

Where is the bug?

There should have been an else clause for \( x \geq 0 \) before this statement.
Is Statement Coverage Sufficient?

- Consider a test set $T = \{ t_1 : x = -5 \}$.
- It is adequate with respect to *statement* coverage criterion, but does not reveal the bug.

```c
1 begin
2 int x, z;
3 input (x);
4 if(x<0)
5   x = -x;
6   z = foo-1(x);
7   output(z);
8 end
```

There should have been an *else* clause for $x \geq 0$ before this statement.
Is Decision Coverage Sufficient?

• Consider another test set $T' = \{ t_1: x = -5 \} \quad t_2: x = 3 \}$
• $T'$ is decision adequate, but not $T$.
• Also, $T'$ reveals the bug, but not $T$.

```
1     begin
2     int x, z;
3     input (x);
4     if (x<0)
5     x = -x;
6     z = foo-1(x);
7     output(z);
8     end
```

There should have been an `else` clause for $x \geq 0$ before this statement.

• This example illustrates *how and why decision coverage might help in revealing a bug that is not revealed* by a test set adequate with respect to statement coverage.
Statement Coverage versus Equivalence Class Partitioning
Example: Identify the Type of a Triangle

- A program $P$ takes an input of three integers $a$, $b$ and $c$, and returns the type of the triangle corresponding to three sides of length $a$, $b$, and $c$, respectively.

- Quiz:

  How to generate a test set based on *Equivalence Class Partitioning* to achieve the highest statement coverage possible?
Example: Identify the Type of a Triangle (2)

**Question:** What is the statement coverage of your test set?

```plaintext
read (a, b, c);
class = scalene;
if a = b || b = a
    class = isosceles;
if a*a = b*b + c*c
    class = right;
if a = b && b = c
    class = equilateral;
case class of
    right             : area = b*c / 2;
    equilateral      : area = a*a * sqrt(3)/4;
    otherwise         : s = (a+b+c)/2;
                          area = sqrt(s*(s-a)*(s-b)*(s-c));
end;
write(class, area);
```
Decision Coverage versus Boundary Value Analysis
**Complement between BVA and Decision Coverage**

- Test cases generated based on Boundary Value Analysis improve decision coverage.

- Similarly, test cases that achieve high decision coverage also cover some boundary values.

- Examples
  - If \( x \leq 0 \) {…..}
    - BVA: \( x_1 = 0; x_2 = 1; x_3 = -1 \)
    - Together, \( x_1, x_2 \) and \( x_3 \) give 100% decision coverage

  - If \( y = 3 \) {…..}
    - \( \{y_1 = 3 \text{ and } y_2 = \text{a value different from } 3\} \text{ gives 100% decision coverage} \)
    - At least one of the boundary values (\( y = 3 \)) is covered
More Examples
χSuds: A Coverage Testing Tool for C Programs