

**SE 2AA4, CS 2ME3 (Introduction to Software
Development)**

Winter 2018

32 White Box Testing Continued (Ch. 6)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

April 2, 2018



32 White Box Testing Continued (Ch. 6)

- Administrative details
- White box testing
 - ▶ Edge coverage
 - ▶ Condition coverage
 - ▶ Path coverage

Administrative Details

- Today's slides are partially based on slides by Dr. Wassying
- A4: Due April 9 at 11:59 pm
- Final tutorial (examination review)
 - ▶ No tutorials Monday or Tuesday
 - ▶ Last tutorials on Friday, Apr 6
- Course evaluations
 - ▶ <https://evals.mcmaster.ca>
 - ▶ Start: Tues, Mar 27, 10:00 am
 - ▶ Close: Tues, Apr 10, 11:59 pm
 - ▶ **Your participation is highly valued!**
 - ▶ *Grade bonus for class participation!*
 - ▶ CS 2ME3 C02: 40%
 - ▶ CS 2ME3 C01: 23%
 - ▶ SE 2AA4 C01: 22%

Theoretical Foundations Of Testing: Definitions

- P (program), D (input domain), R (output domain)
 - ▶ $P: D \rightarrow R$ (may be partial)
- Correctness defined by $OR \subseteq D \times R$
 - ▶ P(d) correct if $\langle d, P(d) \rangle \in OR$
 - ▶ P correct if all P(d) are correct
- Failure
 - ▶ P(d) is not correct
 - ▶ May be undefined (error state) or may be the wrong result
- Error (Defect)
 - ▶ Anything that may cause a failure
 - ▶ Typing mistake
 - ▶ Programmer forgot to test "x=0"
- Fault
 - ▶ Incorrect intermediate state entered by program

Definitions Questions

- A test case t is an element of D or R ?
- A test set T is a finite subset of D or R ?
- How would we define whether a test is successful?
- How would we define whether a test set is successful?

Definitions Continued

- Test case t : An element of D
- Test set T : A finite subset of D
- Test is successful if $P(t)$ is correct
- Test set successful if P correct for all t in T

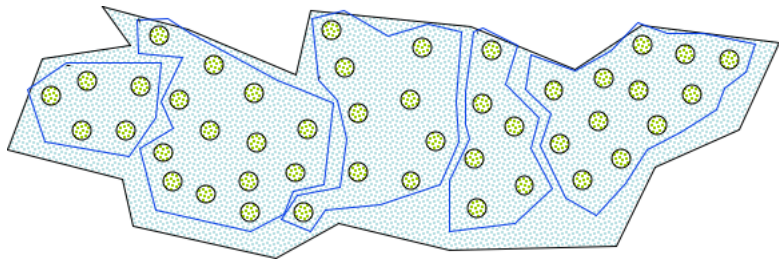
Theoretical Foundations of Testing

- Desire a test set T that is a finite subset of D that will uncover all errors
- Determining an ideal T leads to several **undecidable problems**
- No algorithm exists:
 - ▶ To state if a test set will uncover all possible errors
 - ▶ To derive a test set that would prove program correctness
 - ▶ To determine whether suitable input exists to guarantee execution of a given statement in a given program
 - ▶ etc.

Empirical Testing

- Need to introduce empirical testing principles and heuristics as a compromise between the impossible and the inadequate
- Find a strategy to select **significant** test cases
- Significant means the test cases have a high potential of uncovering the presence of errors

Complete-Coverage Principle



White-box Testing

- Intuitively, after running your test suites, what percentage of the lines of code in your program should be exercised?

White-box Coverage Testing

- (In)adequacy criteria - if significant parts of the program structure are not tested, testing is inadequate
- Control flow coverage criteria
 - ▶ Statement coverage
 - ▶ Edge coverage
 - ▶ Condition coverage
 - ▶ Path coverage

Statement-Coverage Criterion

- Select a test set T such that every elementary statement in P is executed at least once by some d in T
- An input datum executes many statements - try to minimize the number of test cases still preserving the desired coverage

Example

```
read (x); read (y);  
if x > 0 then  
    write ("1");  
else  
    write ("2");  
end if;  
if y > 0 then  
    write ("3");  
else  
    write ("4");  
end if;
```

How would you write a test case?

What is the minimum number of test cases?

Example

```
read (x); read (y);  
if x > 0 then  
    write ("1");  
else  
    write ("2");  
end if;  
if y > 0 then  
    write ("3");  
else  
    write ("4");  
end if;
```

**$\{ \langle x = 2, y = -3 \rangle, \langle x = -13, y = 51 \rangle, \langle x = 97, y = 17 \rangle, \langle x = -1, y = -1 \rangle \}$
covers all statements**

**$\{ \langle x = -13, y = 51 \rangle, \langle x = 2, y = -3 \rangle \}$
is minimal**

Weakness of the Criterion

```
if x < 0 then  
    x := -x;  
end if;  
z := x;
```

$\{<x=-3>\}$ covers all statements. Why is this not enough?

Weakness of the Criterion

```
if x < 0 then  
    x := -x;  
end if;  
z := x;
```

$\{x = -3\}$ covers all
statements

it does not exercise the
case when x is positive
and the then branch is
not entered

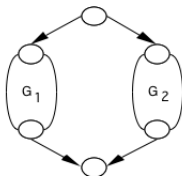
Edge-Coverage Criterion

- Select a test set T such that every edge (branch) of the control flow is exercised at least once by some d in T
- This requires formalizing the concept of the control graph and how to construct it
 - ▶ Edges represent statements
 - ▶ Nodes at the ends of an edge represent entry into the statement and exit

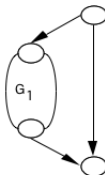
Control Graph Construction Rules



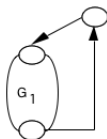
I/O, assignment,
or procedure call



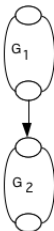
if-then-else



if-then



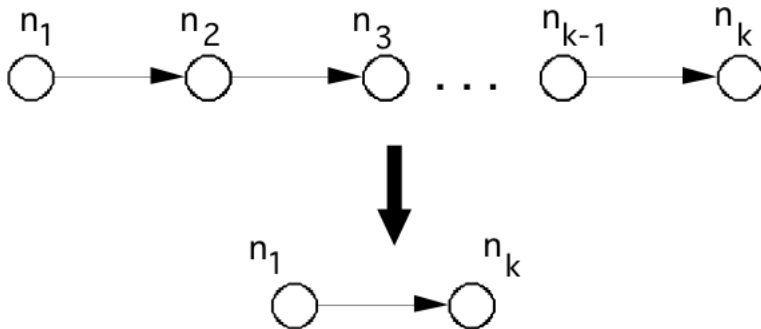
while loop



two sequential
statements

Simplification

A sequence of edges can be collapsed into just one edge



Example: Euclid's Algorithm

```
begin
  read (x); read (y);
  while  $x \neq y$  loop
    if  $x > y$  then
       $x := x - y$ ;
    else
       $y := y - x$ ;
    end if;
  end loop;
  gcd := x;
end;
```

Draw the control
flow graph

Example: Euclid's Algorithm

begin

 read (x); read (y);

 while $x \neq y$ loop

 if $x > y$ then

$x := x - y$;

 else

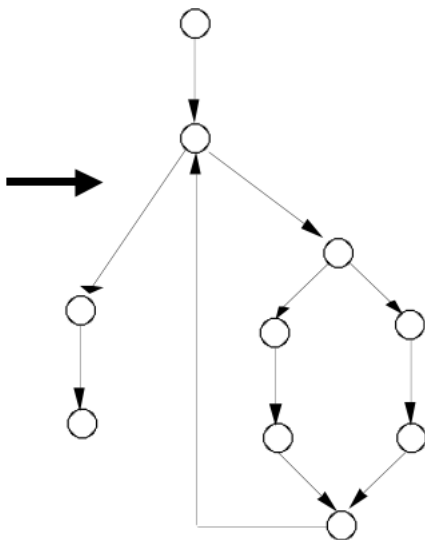
$y := y - x$;

 end if;

 end loop;

 gcd := x;

end;



Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

First list index is 1 (not 0)

What is the problem?

Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

First list index is 1 (not 0)

What is the problem?

Do not discover the error ($<$ instead of \leq)

```
if c1 and c2 then  
    st;  
else  
    sf;
```

// equivalent to

```
if c1 then  
    if c2 then  
        st;  
    else  
        sf;  
else  
    sf;
```


Condition-Coverage Criterion

- Select a test set T such that every edge of P 's control flow is traversed and all possible values of the constituents of compound conditions are exercised at least once
- This criterion is finer than edge coverage

Weakness

```
if  $x \neq 0$  then
     $y := 5$ ;
else
     $z := z - x$ ;
end if;
if  $z > 1$  then
     $z := z / x$ ;
else
     $z := 0$ ;
end if;
```

$\{ \langle x = 0, z = 1 \rangle, \langle x = 1, z = 3 \rangle \}$
causes the execution of all edges,
but fails to expose the risk of a
division by zero

Path-Coverage Criterion

- Select a test set T that traverses all paths from the initial to the final node of P 's control flow
- It is finer than the previous kinds of coverage
- However, number of paths may be too large, or even infinite (see while loops)
- Loops
 - ▶ Zero times (or minimum number of times)
 - ▶ Maximum times
 - ▶ Average number of times

The Infeasibility Problem

- Syntactically indicated behaviours (statements, edges, etc.) are often impossible
- Unreachable code, infeasible edges, paths, etc.
- Adequacy criteria may be impossible to satisfy
 - ▶ Manual justification for omitting each impossible test case
 - ▶ Adequacy “scores” based on coverage - example 95 % statement coverage

Further Problem

- What if the code omits the implementation of some part of the specification?
- White box test cases derived from the code will ignore that part of the specification!