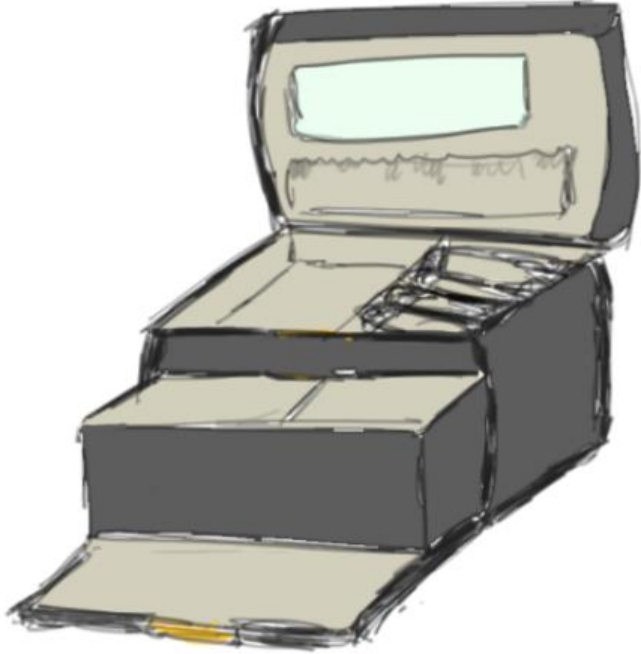CS3300 Introduction to Software Engineering

Lecture 17: White-Box Testing
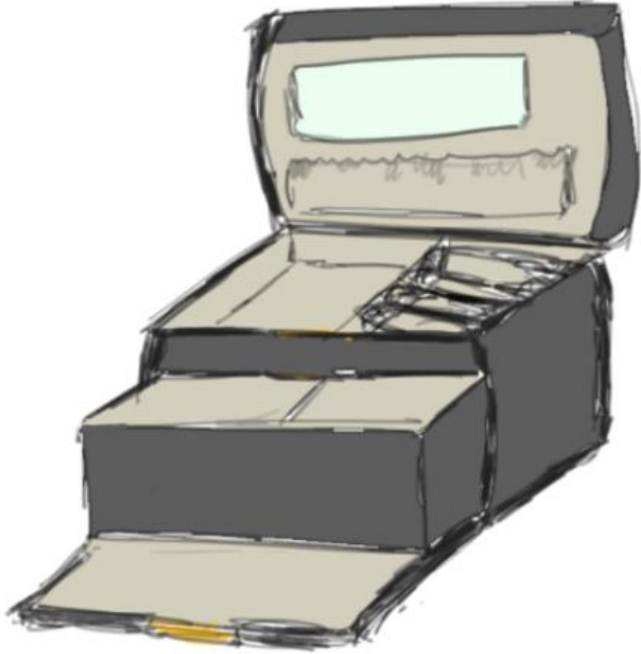
Nimisha Roy ▸ nroy9@gatech.edu

White- Box Testing

Basic Assumption

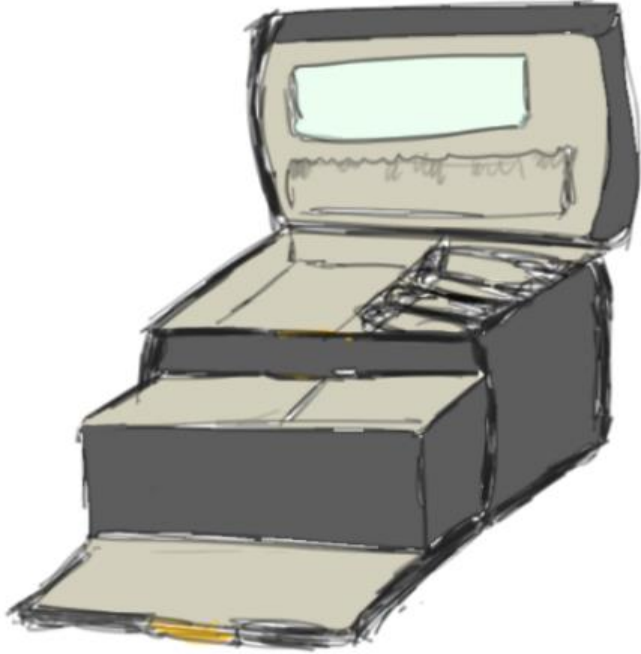Executing the faulty statement is a necessary condition for revealing a fault

White- Box Testing

Advantages

- Based on the code
    - Can be measured objectively
    - Can be measured automatically
- Can be used to compare test suites
- Allows for covering the coded behavior

White- Box Testing

Different Kinds

- <u>Control-Flow Based</u>
- Data-flow based
- Fault based

Let's Consider Program printSum() Again

1. printSum (int a, int b) {
2.   int result = a+b;
3.   if (result > 0)
4.   printcol("red", result);
5.   else if (result < 0)
6.   printcol("blue", result);
7. }

Slide adapted from Alessandro Orso

Coverage Criteria

Defined in terms of

Test requirements - Elements/entities in the code that we need to execute

Result in

Test specifications
Test cases

printSum: Test Requirements

```
1.  printSum (int a, int b) {
2.      int result = a+b;
3.      if (result > 0)
4.          printcol("red", result);
5.      else if (result < 0)
6.          printcol("blue", result);
7.  }
```

Req #1

Req #2

Slide adapted from Alessandro Orso

printSum: Test Specifications

1. printSum (int a, int b) {
2.   int result = a+b;
3.   if (result > 0)
4.   printcol("red", result);
5.   else if (result < 0)
6.   printcol("blue", result);
7. }

Test Spec #1
a + b > 0

Test Spec #2
a + b < 0

printSum: Test Cases

```
1.  printSum (int a, int b) {
2.     int result = a+b;
3.     if (result > 0)
4.         printcol("red", result);
5.     else if (result < 0)
6.         printcol("blue", result);
7.  }
```

Test Spec #1
a + b > 0

Test Spec #2
a + b < 0

#1 ((a = [5], b = [-4]), (output color = [red], output value = [1]

#2 ((a = [0], b = [-1]), (output color = [blue], output value = [-1]

Slide adapted from Alessandro Orso

Coverage Criteria: Statement Coverage

Test Requirements	Statements in the program

Coverage Measure	$\dfrac{\text{Number of executed Statements}}{\text{Total number of Statements}}$

printSum: statement coverage

TC #1
a == 5
b == -4

1. printSum (int a, int b) {
2.   int result = a+b;
3.   if (result > 0)
4.   printcol("red", result);
5.   else if (result < 0)
6.   printcol("blue", result);
7. }

Coverage: 0%

printSum: statement coverage

TC #1
a == 5
b == -4

1. printSum (int a, int b) {
2.   int result = a+b;
3.   if (result > 0)
4.   printcol("red", result);
5.   else if (result < 0)
6.   printcol("blue", result);
7. }

Coverage: 71%

printSum: statement coverage

TC #1
a == 5
b == -4

TC #2
a == 0
b == -1

```
1.  printSum (int a, int b) {
2.      int result = a+b;
3.      if (result > 0)
4.          printcol("red", result);
5.      else if (result < 0)
6.          printcol("blue", result);
7.  }
```

Coverage: 100%

Statement coverage in Practice

Most used in Industry

"Typical coverage" target is 80 – 90%

Why don't we aim at 100%

[Unreachable code, dead code, complex sequences,   ]
[Not enough resources   ]

printSum: statement coverage

TC #1
a == 3
b == 9

TC #2
a == -5
b == -8

1. printSum (int a, int b) {
2.   int result = a+b;
3.   if (result > 0)
4.   printcol("red", result);
5.   else if (result < 0)
6.   printcol("blue", result);
7. [else do nothing]
8. }

Coverage is never 100%

Control Flow Graphs

Representation for the code that is very convenient when we run our reason about the code and its structure.
Represents statement with nodes and the flow of control within the code with edges.


```
1. printSum (int a, int b) {
2.    int result = a+b;
3.    if (result > 0)
4.       printcol("red", result);
5.    else if (result < 0)
6.       printcol("blue", result);
      [else do nothing]
7. }
```
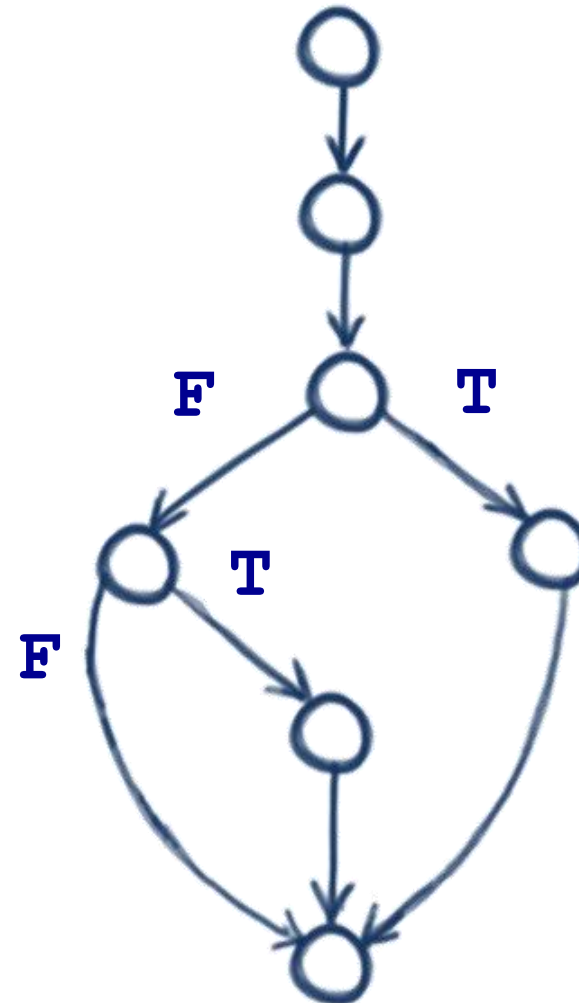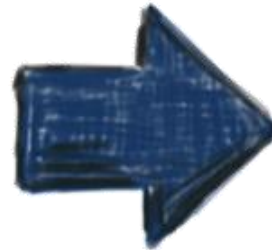
Next Class

Branch Coverage

Condition Coverage

Branch & Condition Coverage

Modified Condition/Decision Coverage

Test Criteria Subsumption

Industry Standards