Software Engineering

CS305, Autumn 2020

Week 13

Class Progress…

- When we met last..

  Software Construction

  - **Coding** - Manual and Automatic Approaches, Paradigms, Reviews
  - **Refactoring** – “Make it easy to read, maintain, and improve”, types, demo, dos and don’ts.
  - **Software Verification** – “checking for bugs”
    - Testing is the most popular method. Inspection, Static analysis, and formal proofs are other methods.
    - IEEE terminology of Failure, Fault, Error.
    - JUnit and Demo in Eclipse.

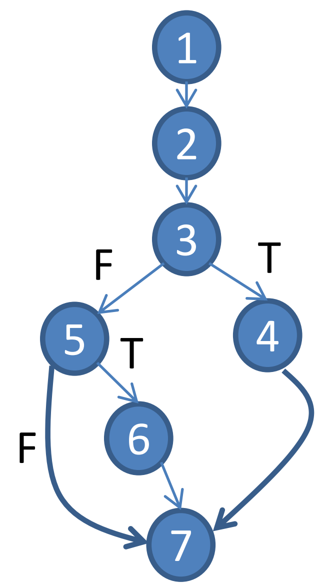
White-Box Testing (contd..) with another example

- White-Box Testing is code based. Hence,
  - Can reveal errors in coding as opposed to Black-Box testing, which deals with observable anomalies (*failure*).
  - Can be objective as opposed to subjective (in Black-Box testing). There are metrics to measure the effectiveness of White-Box testing.
    - Can compare test suites
  - Can be done automatically. There are tools.
- E.g.
  - Code-Coverage based analysis

Code Coverage Based Testing

- Code-coverage based analysis is a *control-flow based approach* (white-box testing can be control-flow based, data-flow based, and fault based)
  - What is control-flow? control-flow graphs (CFGs) to reason about code and structure. E.g.

```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
    cout<<“RED: ”<<result;
else if (result < 0)
    cout<<“BLUE:”<<result;
}
```



Coverage Criteria

```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
     cout<<“RED: ”<<result;
else if (result < 0)
     cout<<“BLUE:”<<result;
}
```

REQ1

REQ2

- Criteria are defined in terms of interesting parts of code that need to be exercised - *test requirements e.g. REQ1, REQ2*
- When you apply a coverage criteria, you get a set of test specifications, test cases.
- E.g. statement coverage, branch coverage.

Assumption: a faulty statement must be executed to uncover a fault

Test Specifications (for REQ1 and REQ2)

```
1. void PrintSum(int a, int b) {
2. int result = a + b;
3. if(result > 0)
4.     cout<<“RED: ”<<result;
5. else if (result < 0)
6.     cout<<“BLUE:”<<result;
7. }
```

REQ1

REQ2

- *REQ1* = “Execute Statement 4”
  - Expressed as constraints on inputs = *“a+b > 0”*

Test Spec 1

- *REQ2* = “Execute Statement 5”
  - Expressed as constraints on inputs = *“a+b < 0”*

Test Spec 2

Implementation of Test Specifications

(for Test Spec 1 and Test Spec 2)

```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
    cout<<“RED: ”<<result;
else if (result < 0)
    cout<<“BLUE:”<<result;
}
```

Test Spec 1

- “a+b > 0”

Input: (a=10, b=10), Expected Output: “RED: 20”

- “a+b < 0”

Test Spec 2

Input: (a=-10, b=-10), Expected Output: “BLUE: -20”

Statement Coverage

- Test Requirement – every statement in the program
- Coverage metric - $\dfrac{\text{number of statements executed}}{\text{Total number of statements}}$

  (higher the ratio better is the coverage)

Statement coverage is satisfied when “all” the statements have been exercised/executed

Can satisfy to different degrees.

- Most used in the industry

Coverage for Test Case 1

```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
    cout<<“RED: ”<<result;
else if (result < 0)
    cout<<“BLUE:”<<result;
}
```

Test Spec 1 is implemented by..

- Test Case 1: `Input: (a=10, b=10), Expected Output: “RED: 20”`

- Coverage: ~71%

Coverage for Test Case 2

```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
     cout<<“RED: ”<<result;
else if (result < 0)
     cout<<“BLUE:”<<result;
}
```

Test Spec 2 is implemented by..

- `Test Case 2:` `Input: (a=-10, b=-10),` `Expected Output: “BLUE: -20”`
- Coverage: ~86%
- Test Case 1 + Test Case 2 = 100% Coverage

Often the expected statement coverage is set to < 100%. Why?

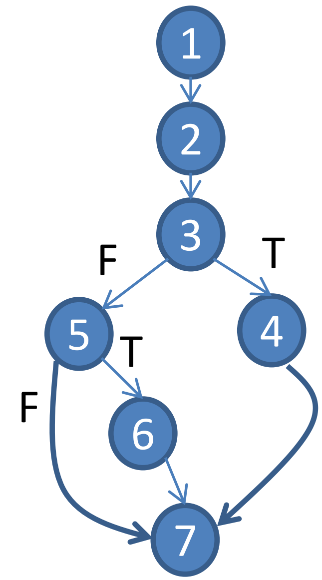
Branch Coverage

- Another type of coverage criteria
- Test Requirement – every branch in the program
- Coverage metric - $\dfrac{\text{number of branches executed}}{\text{Total number of branches}}$

  (higher the ratio better is the coverage)
- A branch = outgoing edges from a decision point in a CFG

Branch Coverage - Example

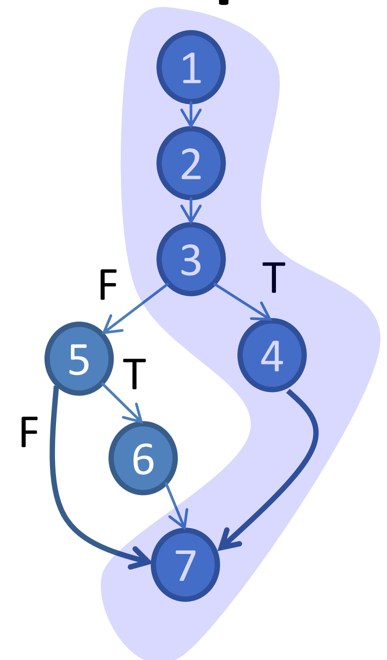
```
1. void PrintSum(int a, int b) {
2. int result = a + b;
3. if(result > 0)
4.     cout<<"RED: "<<result;
5. else if (result < 0)
6.     cout<<"BLUE:"<<result;
7. }
```



- 4 outgoing edges. So, 4 branches.

Branch Coverage - Example

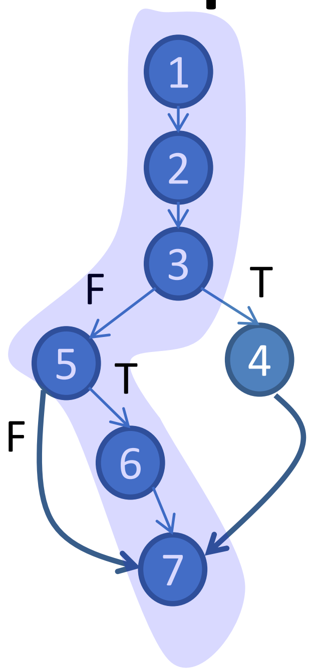
```
1. void PrintSum(int a, int b) {
2. int result = a + b;
3. if(result > 0)
4.      cout<<“RED: ”<<result;
5. else if (result < 0)
6.      cout<<“BLUE:”<<result;
7. }
```



- 4 outgoing edges. So, 4 branches.
- Test case 1: `Input: (a=10, b=10), Expected Output: “RED: 20”`
  - Coverage = 25%

Branch Coverage - Example

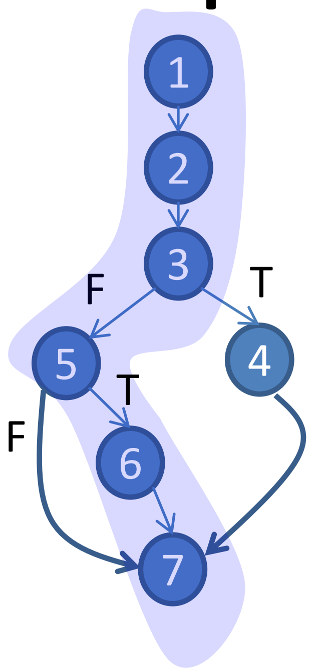
```
void PrintSum(int a, int b) {
int result = a + b;
if(result > 0)
      cout<<"RED: "<<result;
else if (result < 0)
      cout<<"BLUE:"<<result;
}
```



- Test case 2: `Input: (a=-10, b=-10), Expected Output: "BLUE: -20"`
  - Coverage = 50%
- Test case 1 + Test case 2 = 75% coverage (not correct to sum the coverage of individual tests i.e. coverage(Test1) + coverage(Test2) != coverage(Test1+Test2))

Branch Coverage - Example

```
1. void PrintSum(int a, int b) {
2. int result = a + b;
3. if(result > 0)
4.     cout<<"RED: "<<result;
5. else if (result < 0)
6.     cout<<"BLUE:"<<result;
7. }
```



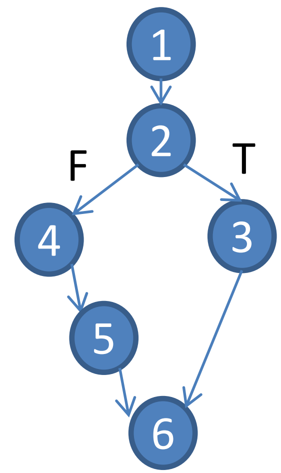
- Test case 3: `Input: (a=0, b=0), Expected Output:`
- Test case 1 + Test case 2 + Test case 3= 100% coverage

Criteria Subsumption

- We tested more thoroughly when moved from statement coverage criteria to branch coverage criteria
  - E.g. we could a test case (a=0,b=0) to go over the F edge of node 5.
- All test cases satisfying a particular criteria also satisfy another criteria. One criteria subsumes another.
  - E.g. all test cases (1-3) yielding 100% branch coverage also yield 100% statement coverage
- Branch coverage is a stronger criteria than statement coverage

Branch Coverage - Example

```
1. void Foo(int x, int y) {
2. if((x==0) || (y>0))
3.     y = y/x; cout<<y;
4. else
5.     x = y + 2; cout<<x;
6. }
```

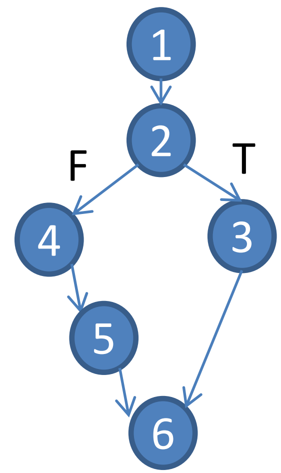


- Test case 1: `Input: (x=-10, y=-10), Expected Output: -8`
- Test case 2: `Input: (x=10, y=10), Expected Output: 1`
- Test case 1 + Test case 2 = 100% branch coverage.

Is 100% branch coverage sufficient to uncover faults?

Branch Coverage - Example

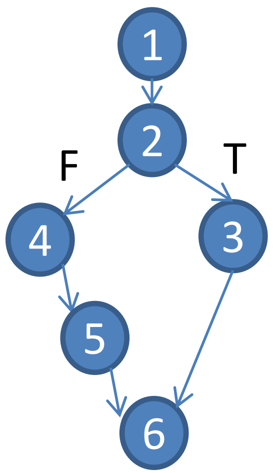
```
void Foo(int x, int y) {
if((x==0) || (y>0))
    y = y/x; cout<<y;
else
    x = y + 2; cout<<x;
}
```



- Test case 3: `Input: (x=0, y=10), Expected Output: divide-by-zero error`
- Instead of considering the whole statement at the decision point (whole predicate), we can consider each of the conditions separately.

Condition Coverage - Example

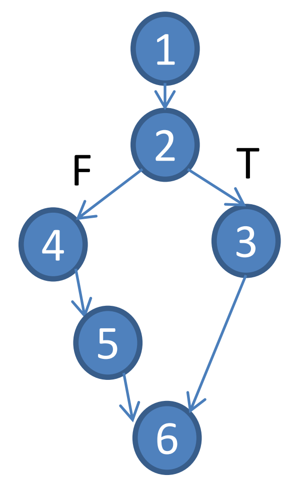
```
1. void Foo(int x, int y) {
2. if((x==0) || (y>0))
3.      y = y/x; cout<<y;
4. else
5.      x = y + 2; cout<<x;
6. }
```



- Test case 1: `Input: (x=0, y=-10), Expected Output: divide-by-zero error`

Condition Coverage - Example

```
void Foo(int x, int y) {
if((x==0) || (y>0))
      y = y/x; cout<<y;
else
      x = y + 2; cout<<x;
}
```



- Test case 2: `Input: (x=-10, y=10), Expected Output:-1`
- Test case 1 + Test case 2 = 100% condition coverage

Does 100% condition coverage mean 100% branch coverage? i.e. Does Condition Coverage subsume Branch Coverage?

Other Coverage Criteria

- Path coverage
- Data-flow coverage
- Mutation coverage

Theoretical. Practically not possible in most cases.

Concluding Remarks

- Any criteria and satisfiability metric is only an approximation for testing
  - Only exhaustive testing can reveal faults
  - E.g. path coverage of 100% in the below code still not able to uncover the fault

```
void Foo(int x, int y) {
int i;
read(i);
print(10/(i-3));
}
```

- Watch out for unreachable/dead code