

SOFTWARE TESTING

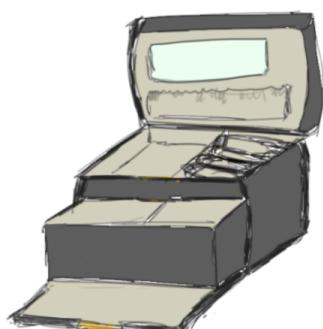
WHITE-BOX TESTING

Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

WHITE-BOX TESTING

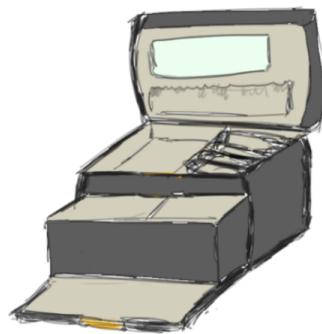
Basic assumption

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



WHITE-BOX TESTING

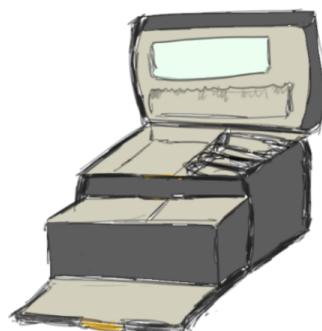
Advantages



- 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 :



- control-flow based
- data-flow based
- fault based

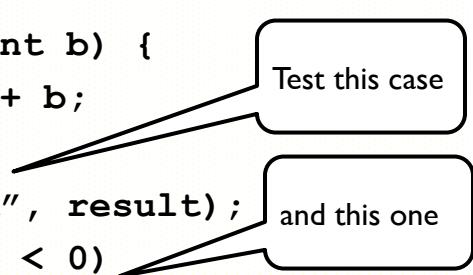
Let's go back to printSum

```
printSum(int a, int b)
```

Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

Let's go back to printSum

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```



Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

COVERAGE CRITERIA

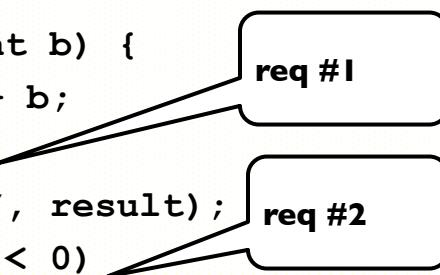
Defined in terms of
test requirements

Result in
test specifications
test cases

Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

printSum: test requirements

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```



printSum: test specifications

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0) a + b > 0  
        printcol("red", result);  
    else if (result < 0) a + b < 0  
        printcol("blue", result);  
}
```



printSum: test cases

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0) Test Spec #1  
a + b > 0  
        printcol("red", result);  
    else if (result < 0) Test Spec #2  
a + b < 0  
        printcol("blue", result);  
}
```

```
#1 ((a=[ ], b=[ ]),(outputColor=[     ], outputValue=[     ]))  
#2 ((a=[ ], b=[ ]),(outputColor=[     ], outputValue=[     ]))
```

printSum: test cases

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

a == 3
b == 9

a == -5
b == -8

STATEMENT COVERAGE

Test requirements

statements in the program

Coverage measure

$\frac{\text{number of executed statements}}{\text{total number of statements}}$

printSum: statement coverage

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

printSum: statement coverage

a == 3
b == 9

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

Coverage: 0%

printSum: statement coverage

a == 3
b == 9

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

Coverage: 71%

printSum: statement coverage

a == 3 a == -5
b == 9 b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

Coverage: 71%

printSum: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

Coverage: 100%

STATEMENT COVERAGE IN PRACTICE

Most used in industry

"Typical coverage" target is 80-90%.



Why don't we aim at 100%?

[

]

printSum: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
}
```

Coverage: 100%

printSum: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing]  
}
```

Coverage: 100%

printSum: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing]  
}
```

Coverage: 100%

printSum: statement coverage

a == 3
b == 9

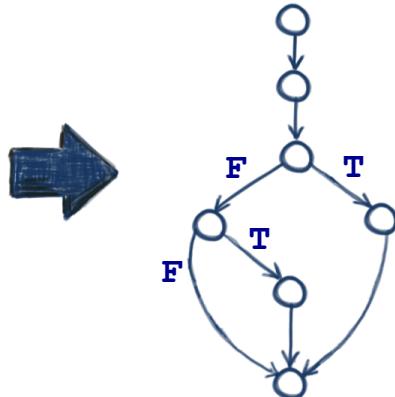
a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing]  
}
```

Coverage: 100%

DIGRESSION : CONTROL FLOW GRAPHS

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



BRANCH COVERAGE

Test requirements

branches in the program

Coverage measure

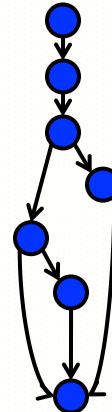
$\frac{\text{number of executed branches}}{\text{total number of branches}}$

printSum: branch coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing] }
```



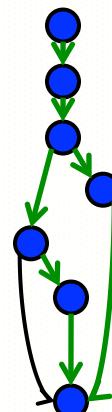
Coverage: ?

printSum: branch coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing] }
```

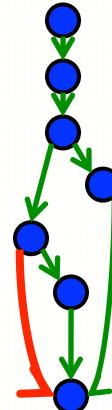


Coverage: 75%

printSum: branch coverage

a == 3 a == -5 a == 0
b == 9 b == -8 b == 0

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing]}
```

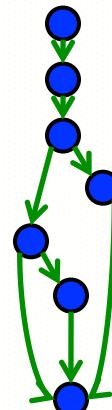


Coverage: 75%

printSum: branch coverage

a == 3 a == -5 a == 0
b == 9 b == -8 b == 0

```
printSum(int a, int b) {  
    int result = a + b;  
    if (result > 0)  
        printcol("red", result);  
    else if (result < 0)  
        printcol("blue", result);  
    [else do nothing]}
```



Coverage: 100%

TEST CRITERIA SUBSUMPTION

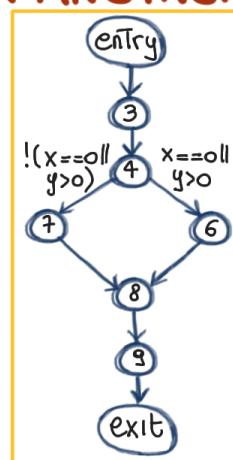


LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){
2.     float x,y;
3.     read(x);
4.     read(y);
5.     if ((x==0)|| (y>0))
6.         y=y/x;
7.     else x=x+2;
8.     write(x);
9.     write(y);
10. }
```

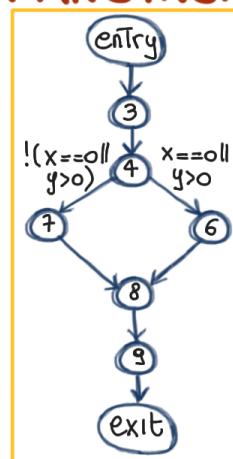
LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){
2.     float x,y;
3.     read(x);
4.     read(y);
5.     if ((x==0) || (y>0))
6.         y = y/x;
7.     else x = y+2;
8.     write(x);
9.     write(y);
10. }
```



LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){
2.     float x,y;
3.     read(x);
4.     read(y);
5.     if ((x==0) || (y>0))
6.         y = y/x;
7.     else x = y+2;
8.     write(x);
9.     write(y);
10. }
```

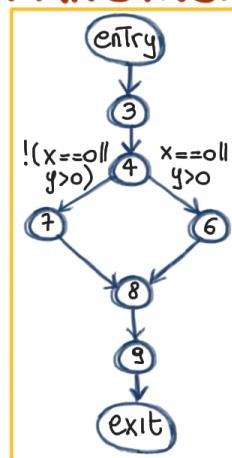


Tests: $(x=5, y=6)$
 $(x=5, y=-5)$

Branch coverage:

LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){  
2.     float x,y;  
3.     read(x);  
4.     read(y);  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```

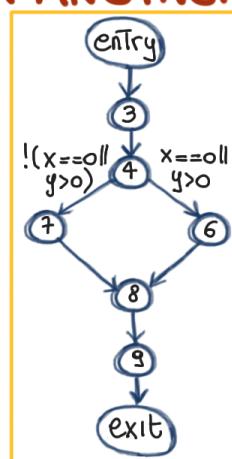


Tests: $(x=5, y=6)$
 $(x=5, y=-5)$

Branch coverage: 100%.

LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){  
2.     float x,y;  
3.     read(x);  
4.     read(y);  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```

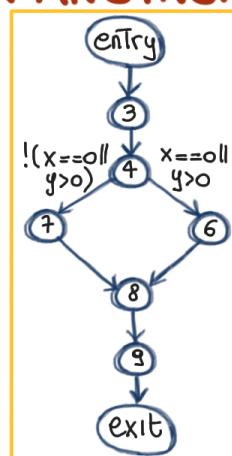


Tests: $(x=5, y=6)$
 $(x=5, y=-5)$

Branch coverage: 100%.
How can we be more thorough?

LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){  
2.     float x,y;  
3.     read(x);  
4.     read(y);  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```



Tests: $(x=5, y=5)$
 $(x=5, y=-5)$

Branch coverage: 100%.
How can we be more thorough?

We can make each condition T and F

CONDITION COVERAGE

Test requirements

individual conditions in the program

Coverage measure

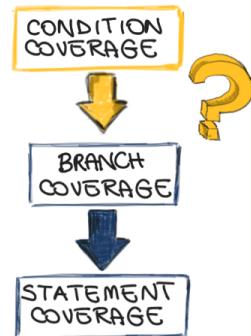
$$\frac{\text{number of conditions that are both T and F}}{\text{total number of conditions}}$$



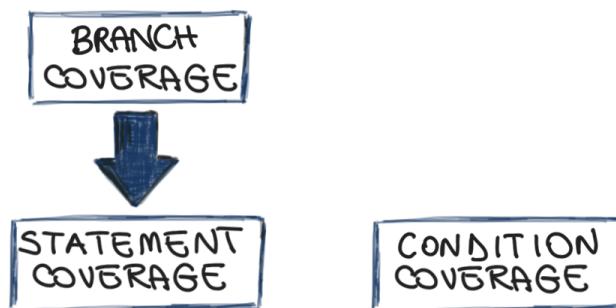
SUBSUMPTION

Does condition coverage imply branch coverage?

- Yes
- No

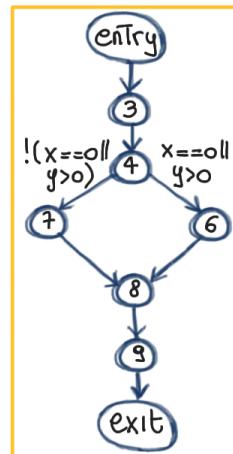


TEST CRITERIA SUBSUMPTION



LET'S CONSIDER OUR LAST EXAMPLE

```
1. void main(){  
2.     float x,y;  
3.     read(x);  
4.     read(y);  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```



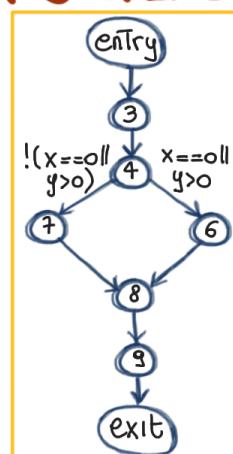
Tests: $(x=0, y=-5)$
 $(x=5, y=5)$

Condition coverage: 100%

What about branch
coverage?

LET'S CONSIDER OUR LAST EXAMPLE

```
1. void main(){  
2.     float x,y;  
3.     read(x);  
4.     read(y);  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```



Tests: $(x=0, y=-5)$
 $(x=5, y=5)$

Condition coverage: 100%

What about branch
coverage? 50%.

BRANCH AND CONDITION COVERAGE (DECISION)

Test requirements

branches and individual conditions in the program

Coverage measure

Computed considering both coverage measures

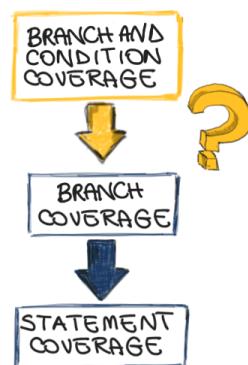


SUBSUMPTION

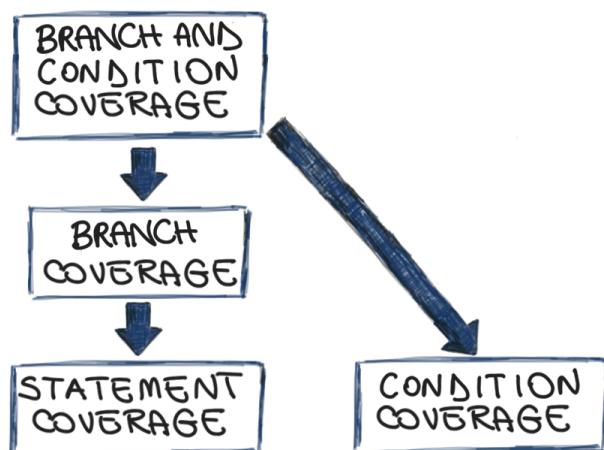
Does branch and condition coverage imply branch coverage?

Yes

No

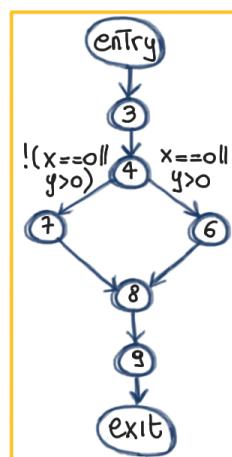


TEST CRITERIA SUBSUMPTION



QUIZ ACHIEVING 100% B&C COVERAGE

```
1. void main(){  
2.     float x,y;  
3.     read x;  
4.     read y;  
5.     if ((x==0) || (y>0))  
6.         y = y/x;  
7.     else x = y+2;  
8.     write(x);  
9.     write(y);  
10. }
```



Test cases
($x=0, y=-5$)
($x=5, y=5$)
Add a Test Case
To achieve 100%
B&C Coverage
($x=$ [] , $y=$ [])

MODIFIED CONDITION/DECISION COVERAGE (MC/DC)

Key idea: test important combinations of conditions and limited testing costs

⇒ extend branch and decision coverage with the requirement that each condition should affect the decision outcome independently

MC/DC : EXAMPLE $a \& \& b \& \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

MC/DC : EXAMPLE

$a \& b \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

MC/DC : EXAMPLE

$a \& b \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False

MC/DC : EXAMPLE

$a \& \& b \& \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False

MC/DC : EXAMPLE

$a \& \& b \& \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

MC/DC : EXAMPLE

a && b && C

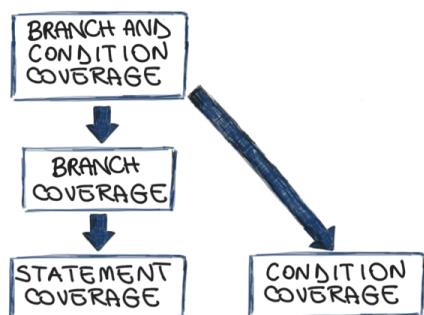
Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

MC/DC : EXAMPLE

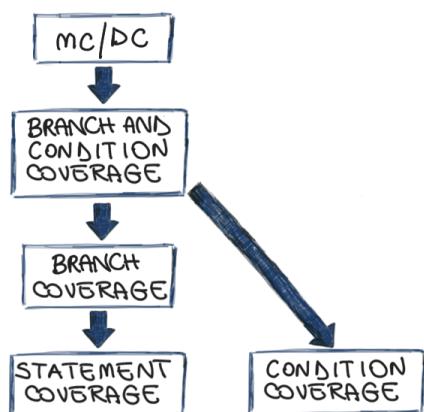
a && b && C

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False
3	True	False	True	False
2	True	True	False	False

TEST CRITERIA SUBSUMPTION



TEST CRITERIA SUBSUMPTION

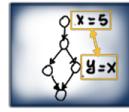


OTHER CRITERIA

OTHER CRITERIA



Path coverage



Data-flow coverage



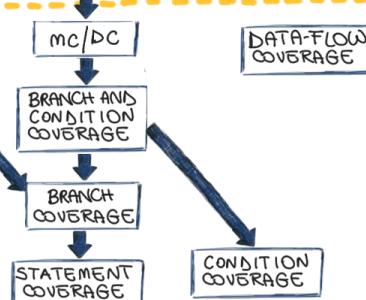
Mutation coverage

TEST CRITERIA SUBSUMPTION

Theoretical
Criteria

PATH COVERAGE MULTIPLE CONDITION COVERAGE MUTATION COVERAGE

Practical
Criteria



1. int i;
2. read(i);
3. print(10/(i-3));

Test suite : (1, -5), (-1, 2.5), (0, -3.3)

Does it achieve path
coverage?

- [] Yes
[] No

Does it reveal the
fault at line 3?

- [] Yes
[] No



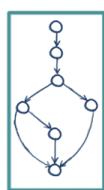
```
1. int i=0;  
2. int j;  
3. read(j);  
4. if ((j>5)&&(i>0))  
5. print(i);
```

Can you create a test suite to achieve statement coverage?

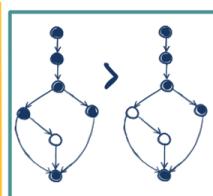
- Yes
 No

.

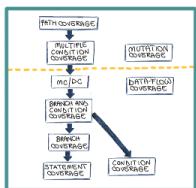
WHITE-BOX TESTING SUMMARY



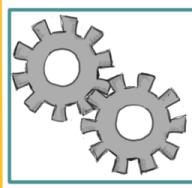
works on a formal model



Comparable



Two broad classes:
practical
Theoretical



Fully automatable