CS 6300 — Class 18

Topics

- Category-partition assignment
- Project 3, Deliverables 2-4
  - Special schedule (see next slide)
- Software Testing – white-box testing

Cat Part Assignment

Due Mar 12, 2014 3:00 pm

Number of resubmissions allowed

Accept Resubmission Until

Status

Grade Scale:

Modified by instructor

Assignment Instructions

To complete this assignment, you ... that we saw in class. Attached y...

- category-partition.pdf: th...
- grep.pdf: a concise specifi...
- cp-example.txt: the TSL fil...

The TSLgenerator tool for Linux, ...

- https://dl.dropbox.com/u...
- https://dl.dropbox.com/u...
- https://dl.dropbox.com/u...
- https://dl.dropbox.com/u...

Submit, as attachments:

1. The TSL file for the grep a...
2. The test specifications ge...

Additional resources for assig...

category-partition.pdf (1 M...

grep.pdf (42 KB; Mar 5, 2014 ...

cp-example.txt (1 KB; Mar 5, ...

Submission

This assignment allows submissio...

Attachments

No attachments yet

SPECIAL ARTICLE

THE CATEGORY-PARTITION METHOD FOR SPECIFYING AND GENERATING FUNCTIONAL TESTS

A method for c... test engineer a... specifications, ... from which test... the tester can ... control the com... specification u...

THOMAS J. OSTRAND and MARC ...

The goal of functional testing of a softwar... find discrepancies between the actual be... implemented system's functions and the ... ior as described in the system's functiona... To achieve this goal requires first, that te... cised for all of the system's functions, an... the tests be designed to maximize the ch... ing errors in the software. Although a pa... method or testing group may emphasize ... other, these two aspects of testing are ma... mentary, and both are necessary for max... tive testing. It is not enough merely to "t... functionality"; the tests must be aimed a... ...nerable parts of the implementation. For ... testing, this implies testing boundary con... cases, error handlers, and cases where in... system environment interact in potentiall... ways. Conversely, it is not enough to writ... error-exposing tests for only some of a sy... tions, or to write tests aimed only at certa... errors or certain characteristics of a speci... plementation.

A preliminary version of this paper appeared as "Metho... of Software Tests," in the Proceedings of the Fifth Annu... Software Quality Conference, Portland, Oregon, October ...

© 1988 ACM 0001-0782/88/0600-0676 $1.50

676 Communications of the ACM

Category-Partition Method Assignment

NAME

grep - search a file fo...

SYNOPSIS

grep <pattern> <...

DESCRIPTION

The grep utility search... lines that contain tha... that contains multiple ... only once.

The pattern is any seq... in the pattern, the enti... quotes ('). To include ... sign must be escaped (... entire pattern in single...

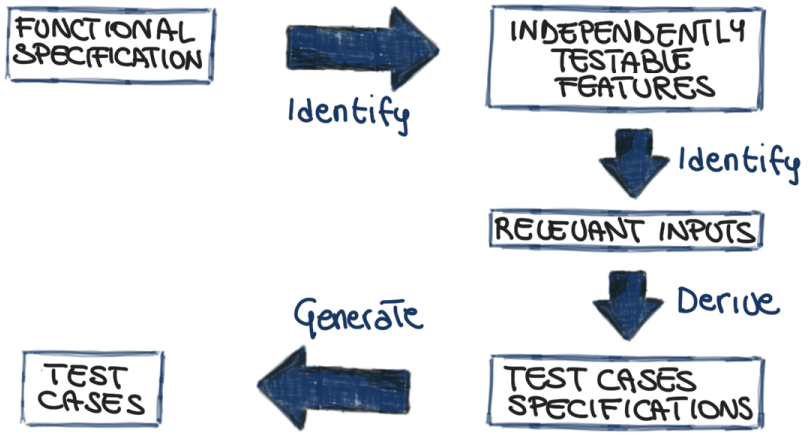
```
# Input string

  Length:
    0.                          [property zerovalue]
    1.
    size - 1.
    size + 1.
    size*2.
    size*2 - 1.
    size*2 + 1.
    maxint.

  Content:
    alphanumeric characters.    [if !zerovalue]
    special characters.         [if !zerovalue]
    spaces.                     [if !zerovalue]

# Input size

  Value:
    0.
    >0.
    <0.                         [error]
    maxint.                     [single]
```

SOFTWARE TESTING

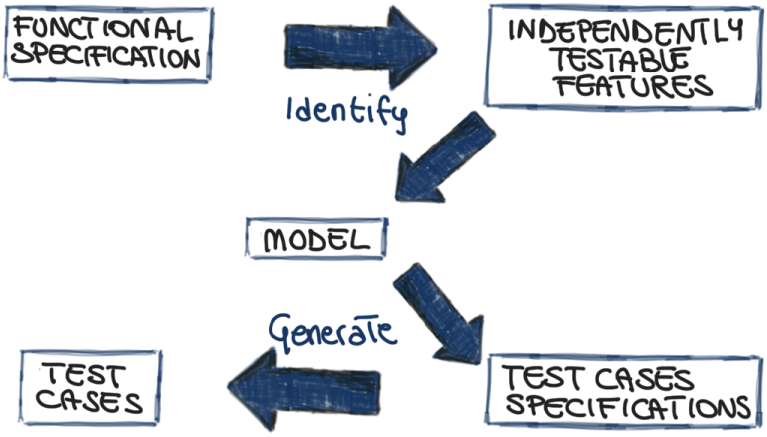
BLACK-BOX TESTING

Wrap up

A SYSTEMATIC FUNCTIONAL-TESTING APPROACH



A SYSTEMATIC FUNCTIONAL-TESTING APPROACH



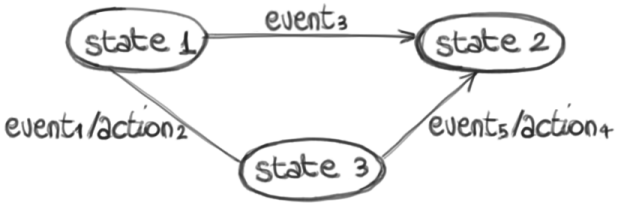
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

MODEL-BASED TESTING

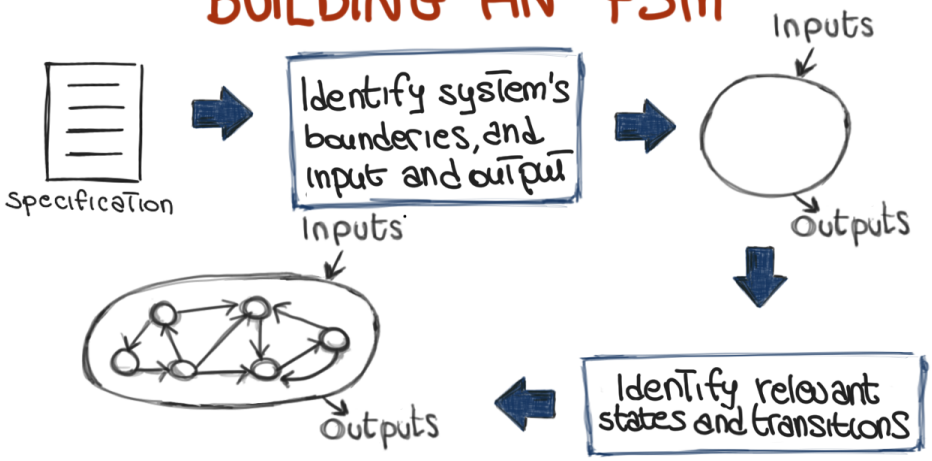
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

FINITE STATE MACHINES (FSM)

Nodes = states
Edges = transitions
Edge labels = events/actions



BUILDING AN FSM



FROM AN INFORMAL SPECIFICATION...

Multiple choices here

Determine the next step

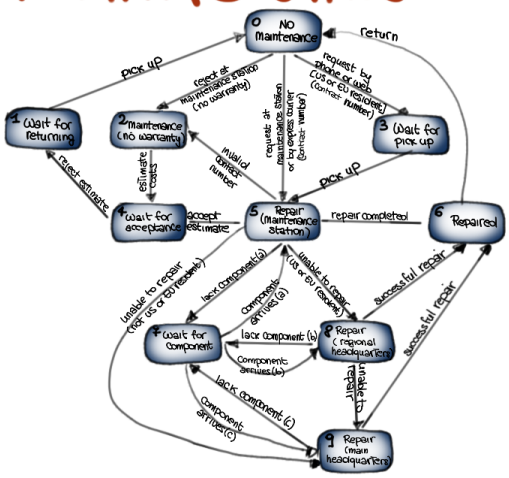
and so on

spec.txt

Maintenance: The Maintenance function records the history of items undergoing maintenance.
If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the web site, or by bringing the item to a designated maintenance station.
If the maintenance is requested by phone or web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.
If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.
If the product is not covered by warranty or maintenance contract, maintenance can be requested only by bringing the item to a maintenance station. The maintenance station informs the customer of the estimated costs for repair.
Maintenance starts only when the customer accepts the estimate.
If the customer does not accept the estimate, the product is returned to the customer.
Small problems can be repaired directly at the maintenance station. If the maintenance station cannot solve the problem, the product is sent to the maintenance regional headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).
If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.
Maintenance is suspended if some components are not available.
Once repaired, the product is returned to the customer.

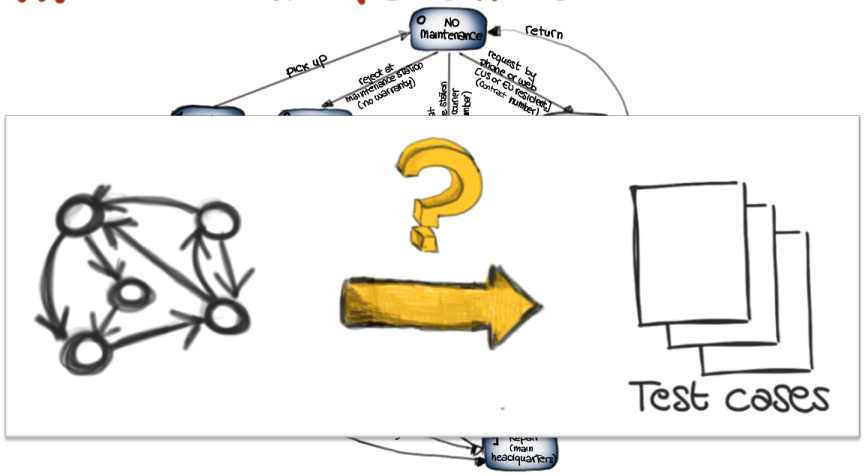
Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

... TO A FINITE STATE MACHINE



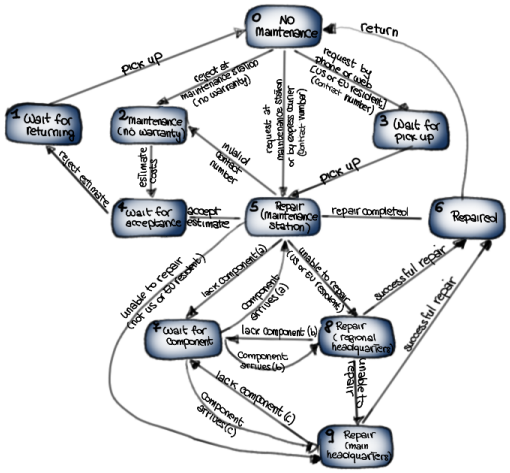
Alex Orso - Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

... TO A FINITE STATE MACHINE



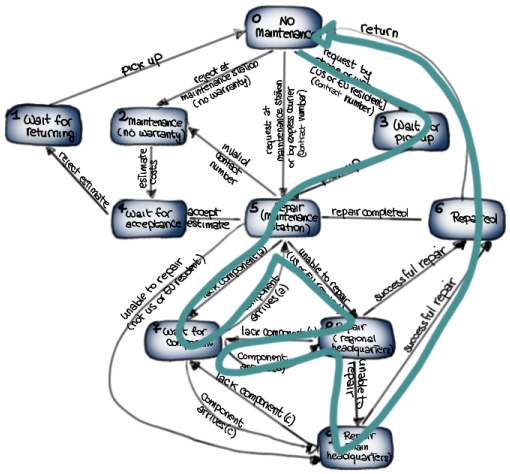
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



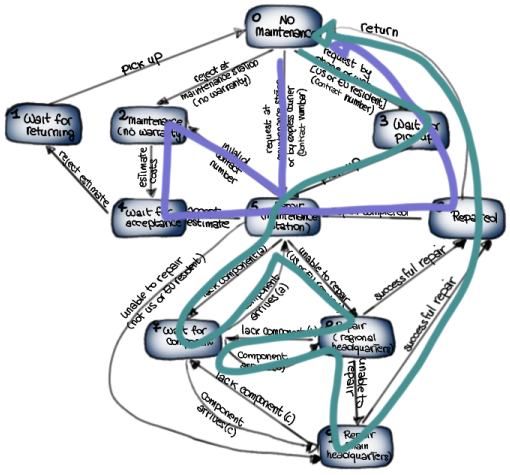
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



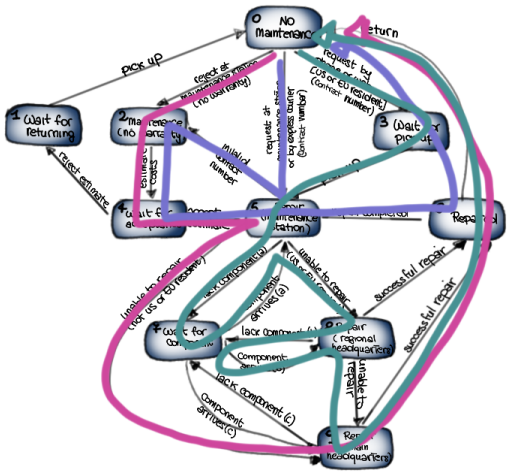
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



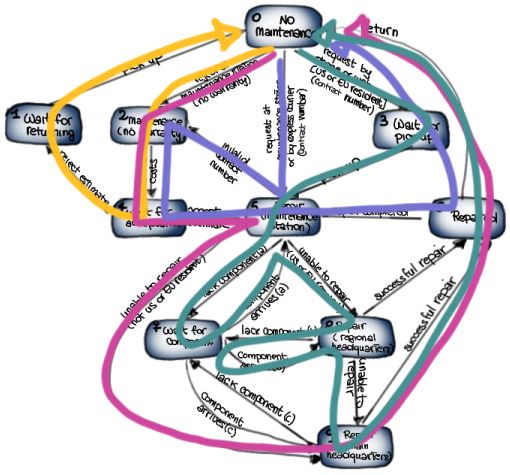
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



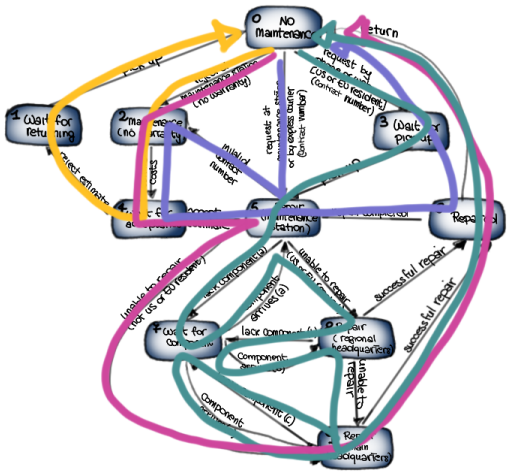
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



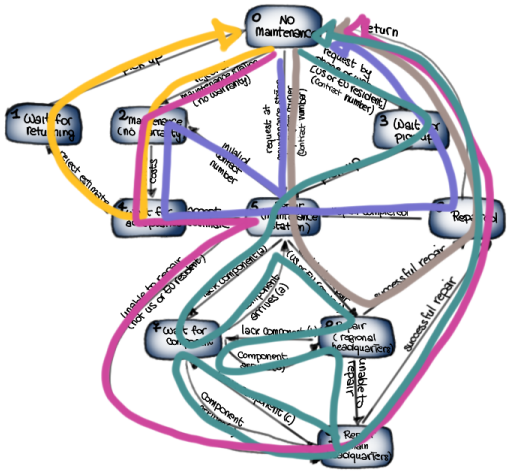
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



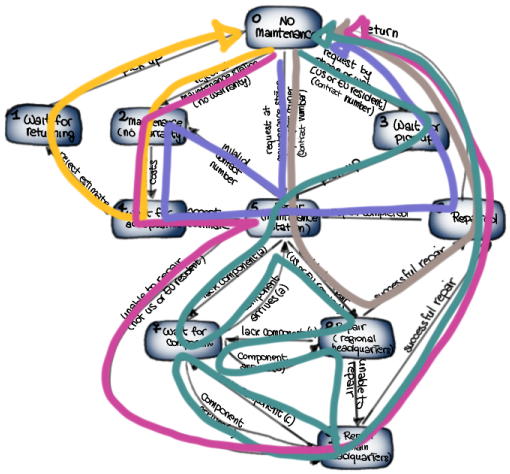
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

TO A SET OF TEST CASES



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

TO A SET OF TEST CASES



TC1: ∅, 3, 5, 7, 5, 8
7, 8, 9, 7, 9, 6, ∅

TC2: ∅, 5, 2, 4, 5, 6, ∅

TC3: ∅, 2, 4, 1, ∅

TC4: ∅, 4, 5, 9, 6, ∅

TC 5 : ∅, 5, 6, ∅

SOME CONSIDERATIONS

Applicability

- very general approach
- in UML, state machine are readily available

Abstraction is key

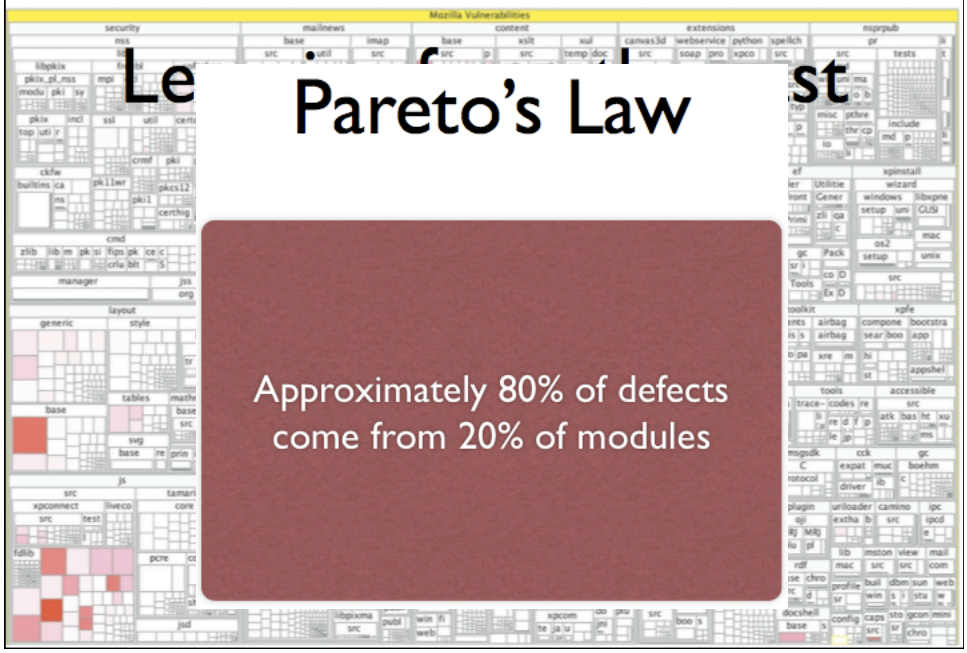
Many other approaches

- decision tables
- flow graphs
- historical models

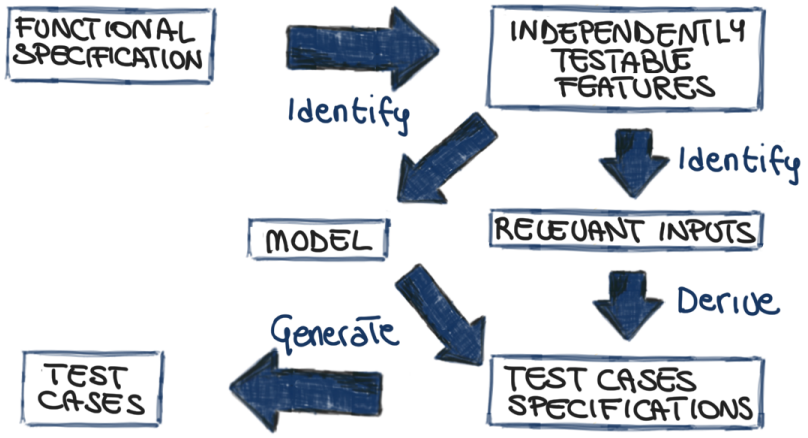
...

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

Historical models



A SYSTEMATIC FUNCTIONAL-TESTING APPROACH



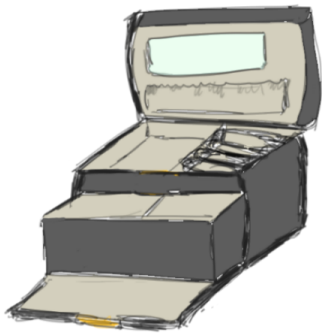
Alex Orso – Partially based on material from Mauro Pezze', Michal Young, and Andreas Zeller

SOFTWARE TESTING

WHITE-BOX TESTING

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

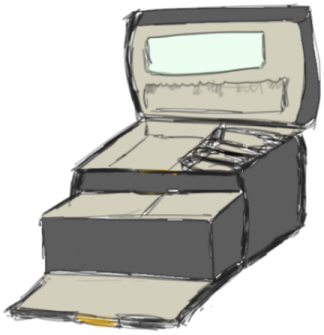
WHITE-BOX TESTING



Basic assumption

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

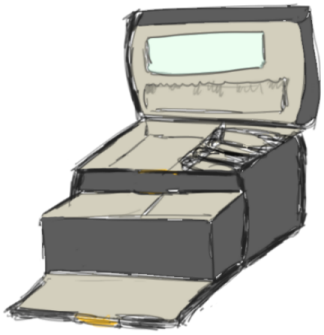
WHITE-BOX TESTING



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)
```


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);
}
```

Test this case

and this one

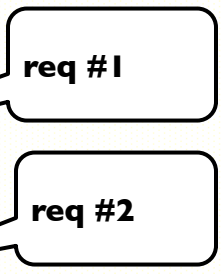
COVERAGE CRITERIA

Defined in terms of
- test requirements

Result in
- test specifications
- test cases

`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)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
}
```

a + b > 0

a + b < 0

`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);
}
```

Test Spec #1
a + b > 0

Test Spec #2
a + b < 0

#1 ((a=[ ], b=[ ]),(output color=[ ], output value=[ ]))

#2 ((a=[ ], b=[ ]),(output color=[ ], output value=[ ]))

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
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: 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

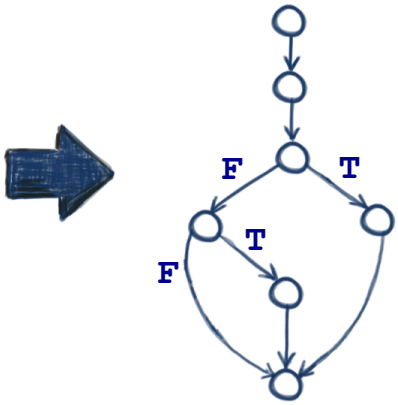
a == 3 b == 9	a == -5 b == -8

```
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.     printCol("red", result);
5.   else if (result<0)
6.     printCol("blue", result);
     [else do nothing]
7. }
```



BRANCH COVERAGE

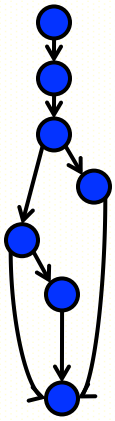
Test requirements: branches in the program

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]}
```

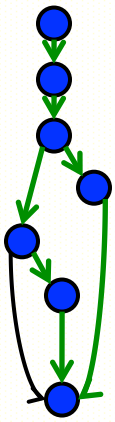


Coverage: ?

`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]}
```

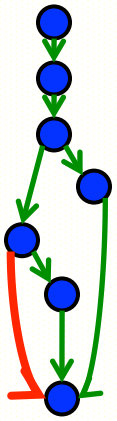


Coverage: 75%

`printSum`: branch coverage

a == 3 b == 9	a == -5 b == -8	a == 0 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]}
```

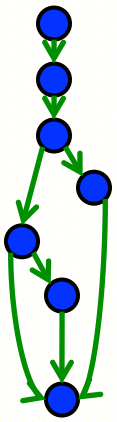


Coverage: 75%

`printSum`: branch coverage

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

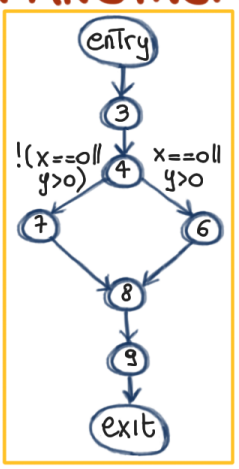
TEST CRITERIA SUBSUMPTION

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=y+2;
8.   write(x);
9.   write(y);
10. }
```

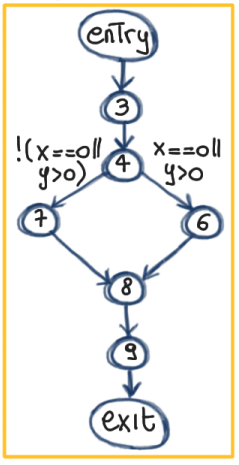
LET'S CONSIDER ANOTHER EXAMPLE

```
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. }
```



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. }
```

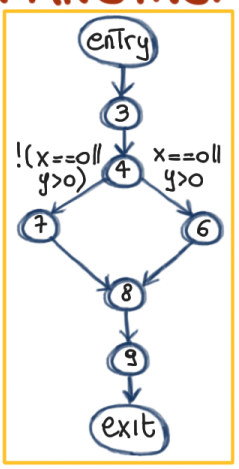


Tests: (x=5, y=6)
(x=5, y=-5)

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. }
```

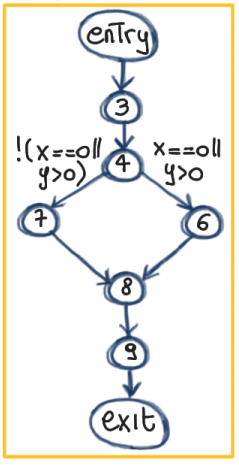


Tests: (x=5, y=6)
(x=5, y=-5)

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. }
```



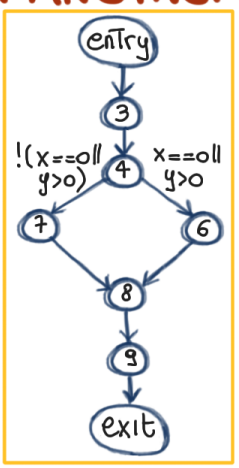
Tests: (x=5, y=6)
(x=5, y=-5)

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=6)
(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

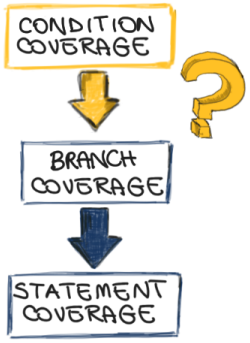
$$\frac{\text{number of conditions that are both T and F}}{\text{total number of conditions}}$$

SUBSUMPTION

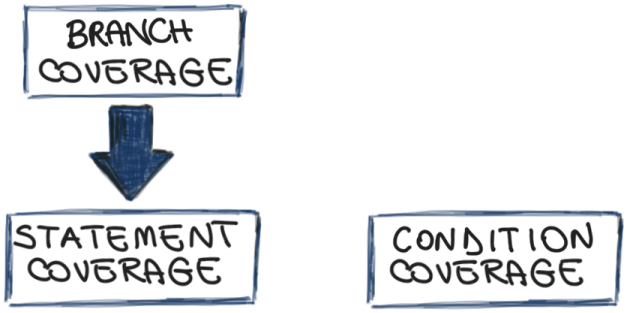
Does condition coverage imply branch coverage?

[ ] Yes

[ ] No

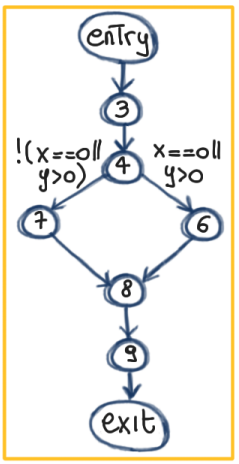


TEST CRITERIA SUBSUMPTION



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. }
```



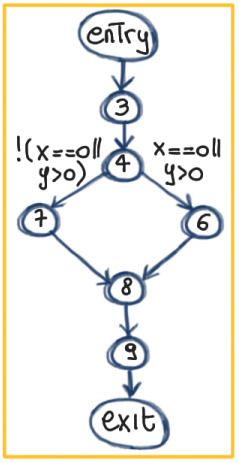
Tests: (x=ø, y=-5)
(x=5, y=5)

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=ø, 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

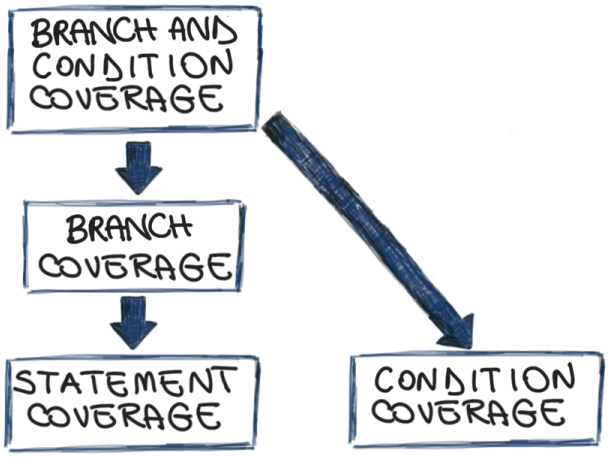
SUBSUMPTION

Does branch and condition coverage imply branch coverage?

[ ] Yes

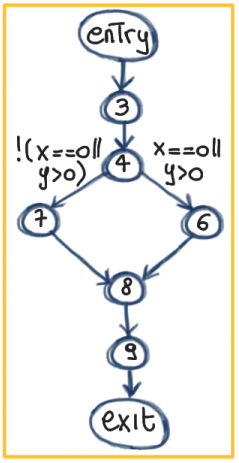
[ ] No

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=∅, 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

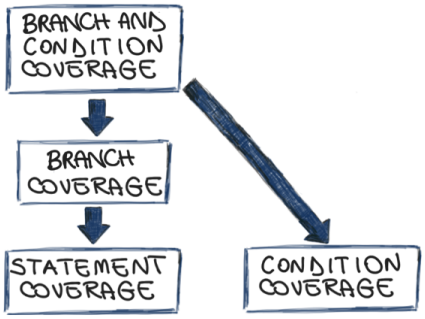
Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

MC/DC : EXAMPLE

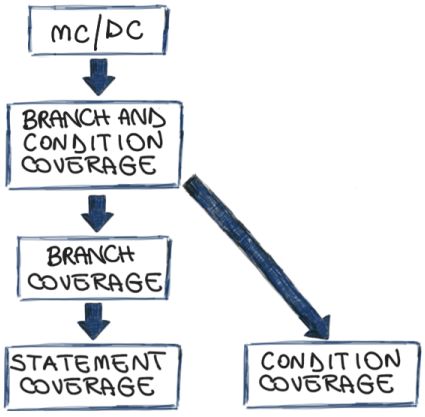
a && b && c

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

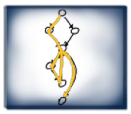


TEST CRITERIA SUBSUMPTION

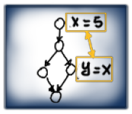


OTHER CRITERIA

OTHER CRITERIA



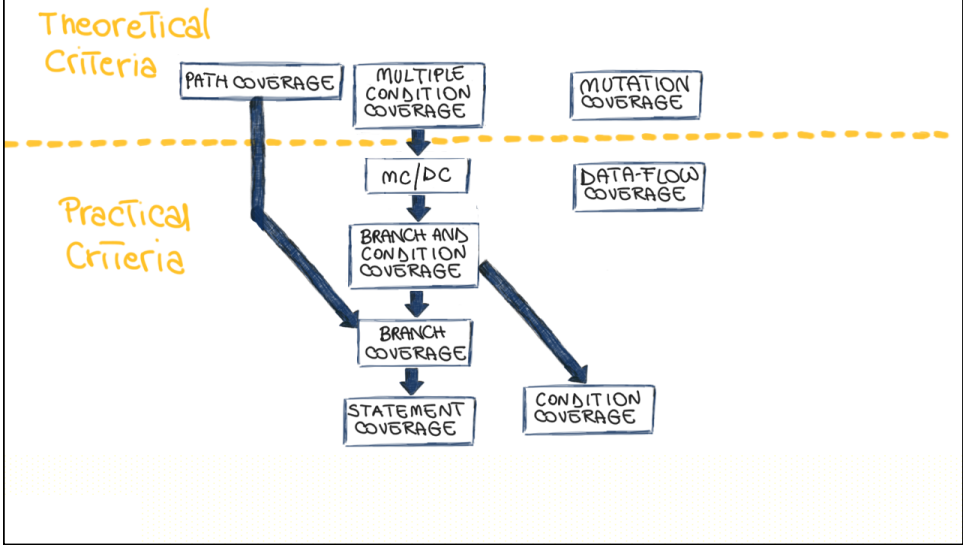
Path coverage



Data-flow coverage

Mutation coverage

TEST CRITERIA SUBSUMPTION




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

Test suite : $(1,-5), (-1,2.5), (0,-3.\bar{3})$

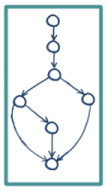
Does it achieve path coverage?	Does it reveal the fault at line 3 ?
[ ] Yes	[ ] Yes
[ ] No	[ ] No


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

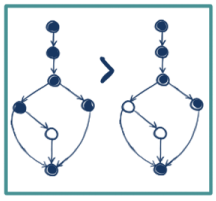
Can you create a test suite to achieve statement coverage?

[ ] Yes
[ ] No

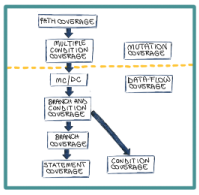
WHITE-BOX TESTING SUMMARY



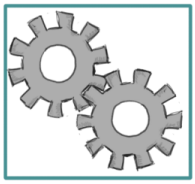
Works on a formal model



Comparable



Two broad classes:
- practical
- Theoretical



Fully automatable