SENG3320/6320:
Software Verification and Validation
Lecture 1

Course Introduction

Problem Motivation

Software Quality Facts

- Only 32% of software projects are considered successful (full featured, on time, on budget)
- Software failures cost the US economy $59.5 billion dollars every year [NIST 2002 Report]
- On average, 1-5 bugs per KLOC (thousand lines of code)
  - In mature software (more than 10 bugs in prototypes)

- 35MLOC
- 63K known bugs at the time of release
- 2 bugs per KLOC

Software fault examples

Intel Pentium Floating-Point Division Bug in 1994:

Enter the following equation into your PC's calculator:
(4195835 / 3145727) * 3145727 – 4195835

- If the answer is zero, your computer is just fine.
- If you get anything else, you have an old Intel Pentium CPU with a floating-point division bug.
- Found by Dr. Thomas R. Nicely in 1994.

https://en.wikipedia.org/wiki/Pentium_FDIV_bug

Software fault examples

- Mars Climate Orbiter (1998)
  - Sent to Mars to relay signal from Mars Lander
  - Smashed to the planet
- Cause: Failing to convert between different metric standards
  - Software that calculated the total impulse presented results in pound-seconds
  - The system using these results expected its inputs to be in newton-seconds

Software fault examples

- USS Yorktown (1997)
  - Left dead in the water for 3 hours
- Cause: Divide by zero error

$$\frac{\text{Number}}{0} =$$

On 21 September 1997, a crew member entered a zero into a database field causing an attempted division by zero in the ship's Remote Data Base Manager, resulting in a buffer overflow which brought down all the machines on the network, causing the ship's propulsion system to fail.

Software fault examples

- ATT (1990)
  - One switching system in New York City experienced an intermittent failure that caused a major service outage
  - The first major network problem in AT&T's 114-year history
- Cause: Wrong BREAK statement in C Code
  - Complete code coverage could have revealed this bug during testing

```
4.      case THING1:
5.          doit1();
6.          break;
7.  case THING2:
8.           if (x == STUFF) {
9.              do_first_stuff();
10.             if (y == OTHER_STUFF)
11.                 break;
12.             do_later_stuff();}
13. /* coder meant to break to here... */
14.          initialize_modes_pointer();
15.          break;
16.    default:
17.         processing(); }
18. /* ...but actually broke to here! */
```

```
                                    /* leaving the m
    odes pointer
```

Software fault examples

- Ariane 5 flight 501 (1996)
  - Destroyed 37 seconds after launch (cost: $370M)
- Cause: Arithmetic overflow
  - Data conversion from a 64-bit floating point to 16-bit signed integer value caused an exception
  - The software from Ariane 4 was re-used for Ariane 5 without re-testing

```
int tryIt () {
int i;
char string[5] = "hello";
for (i=0; i<=5; i++)
   print(string[i]);
}
```

Any bugs
in this program?

```
int minval(int *A, int n) {
  int currmin;

  for (int i=0; i<n; i++)
    if (A[i] > currmin);
      currmin = A[i];
  return currmin;
}
```

Any bugs
in this program?

```
File file = new File("C:/robots.txt");
FileInputStream fis = null;
bool accessGranted = true;
try {
    fis = new FileInputStream(file);
    System.out.println("Total file size to read (in
bytes) : "    + fis.available());
     fis.close();
} catch (SecurityException x) {
    accessGranted = false;    // access denied
} catch (...)  {
    // something else happened
}
```

Any bugs
in this program?

Software Verification and Validation

- Software verification and validation (V&V):
  - Verification: “Are we building the product right”.
    - The software should conform to its specification.
  - Validation: “Are we building the right product”.
    - The software should do what the user really requires.

- V&V Techniques:
  - “Formal” approaches
    - Formal verification
    - Model checking
    - …
  - “Informal” approaches
    - Code Review
    - Static Analysis
    - Software Testing
    - …

Software Testing

"Software testing is the process of executing a program or a system with the intent of finding errors"

--- "The art of software testing" by G. J. Myers

Terms

- **Fault**: incorrect portions of code (may involve missing code as well as incorrect code)
- **Failure**: observable incorrect behavior of a program.
- **Error**: cause of a fault: something bad a programmer did (conceptual, typo, etc)
- **Bug**: informal term for fault

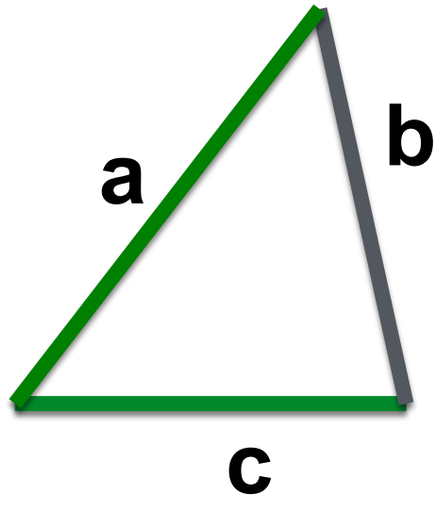
Testing v.s. Debugging

- Testing: Finding inputs that cause the failure of a software
  - Failure is unknown.
  - Performed by “Testers”.

- Debugging: The process of finding and fixing a fault given a failure
  - Failure is known.
  - Performed by “Developers”.

Test Cases

- Test Case: a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement



Test Case

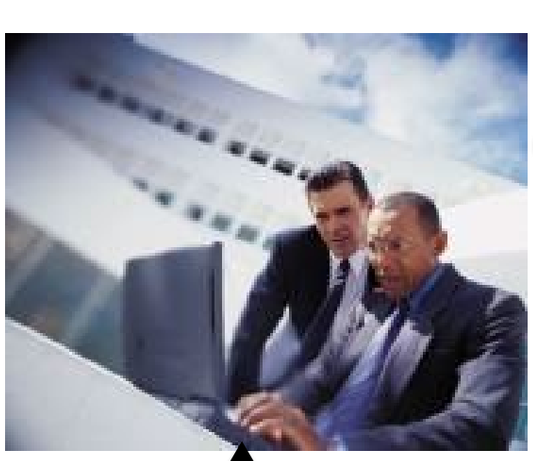
Test Input: a=3, b=4, c=5
Expected Result: right triangle

Test Case

Objective: Gmail-compose mail-able to edit the email
Test Input: 1. Click on compose; 2. Type some text in the message box
Conditions: User should log in Gmail
Expected Result: User is able to type in the message box

A Typical Software Testing Process

Test case generation



Exhaustive Testing is Hard /1

```
int max(int x, int y)
{
   if (x > y)
      return x;
   else
      return x;
}
```

- Number of possible test cases (assuming 32 bit integers)

$$2^{32} \times 2^{32} = 2^{64}$$

- Test suite {(x=3,y=2),(x=4,y=3),(x=5,y=1)} will not detect the error

- Test suite {(x=3,y=2), (x=2,y=3)} will detect the error

- The power of the test suite is not determined by the number of test cases

Exhaustive Testing is Hard /2

- Assume that the input for the `max` procedure was an integer array of size *n*
  - Number of test cases: $2^{32 \times n}$
- Assume that the size of the input array is not bounded
  - Number of test cases: ∞
- The point is, naive exhaustive testing is pretty hopeless

Testing Techniques

- Functional (Black box) vs. Structural (White box) testing
  - Functional testing: Generating test cases based on the functionality of the software
  - Structural testing: Generating test cases based on the structure of the program
  - Black box testing and white box testing are synonyms for functional and structural testing, respectively.
    - In black box testing the internal structure of the program is hidden from the testing process
    - In white box testing internal structure of the program is taken into account

Black-Box Testing

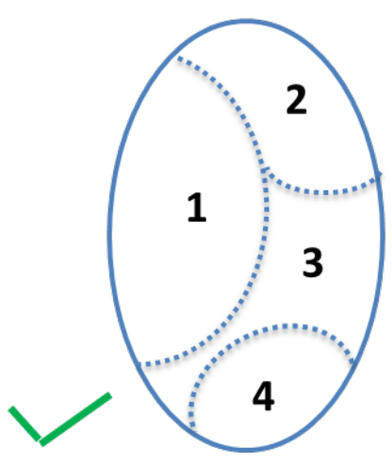
Black-Box Testing

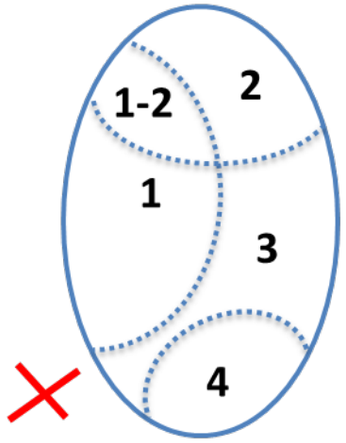
- Identify functions and design test cases that will check whether these functions are correctly performed by the software
    - Formal specifications of the functions
    - Informal specifications (*more popular in industry*)
- Techniques:
    - Equivalence Partitioning
    - Boundary-Value Analysis

Equivalence Partitioning

- Divide the input domain into **equivalence partitions**
- Select one test case from each equivalence partition
- Characteristics of equivalence partitions
  - Disjointedness
    - No input belongs to more than one partition
  - Coverage
    - The input domain is covered by the entire collection of equivalence partitions

Every possible test input belongs to
one and only one partition





Equivalence Partitioning (cont.)

- Let's test a method "isEven(int n)", which

  returns "true" for all even Inputs

  returns "false" for all odd Inputs

  , where $1000 \geq n \geq 1$

- We can create two equivalent partitions, i.e. even numbers and odd numbers between 1 and 1000

Equivalence Partitioning Example

Suppose a Windows application requires a password, which has minimum 8 characters and maximum 12 characters.

Design test cases for the password length checking program.

Equivalence Partitioning Example

Invalid Partition	Valid Partition	Invalid Partition
Less than 8	8 - 12	More than 12

Boundary-Value Analysis (BVA)

Instead of selecting any element in an equivalence partition, inputs close to the boundaries of the equivalence classes are selected as test cases

1 Partition the input domain. This leads to as many partitions as there are input variables. We will generate several sub-domains in this step.

2 Identify the boundaries for each partition. Boundaries may also be identified using special relationships amongst the inputs.

3 Select test data such that each boundary value occurs in at least one test input.

t1: 7
t2: 8
t3: 9
t4: 12
t5: 13

BVA: Example - Create equivalence classes

Assuming that a product code must be in the range 99..999 :

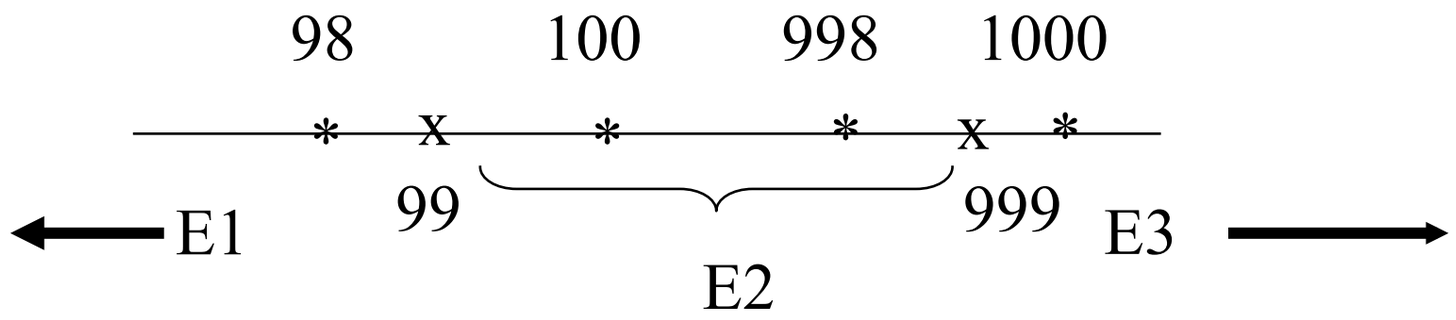
Product Code: [ ]

Equivalence classes for code:

E1: Values less than 99.

E2: Values in the range.

E3: Values greater than 999.

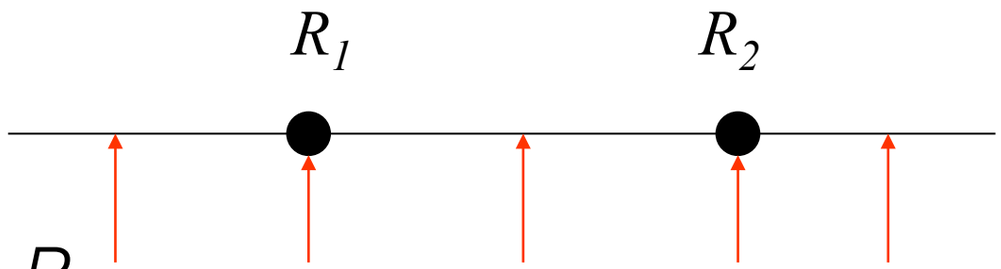


Equivalence classes and boundaries

Boundaries are indicated with an x. Points near the boundary are marked *.

Testing Boundary Conditions

- For each range [R_1, R_2] listed in either the input or output specifications, choose five cases:
  - Values less than R_1
  - Values equal to R_1
  - Values greater than R_1 but less than R_2
  - Values equal to R_2
  - Values greater than R_2
- For unordered sets, select two cases
  - 1) in, 2) not in
- For equality, select two values
  - 1) equal, 2) not equal



Quiz

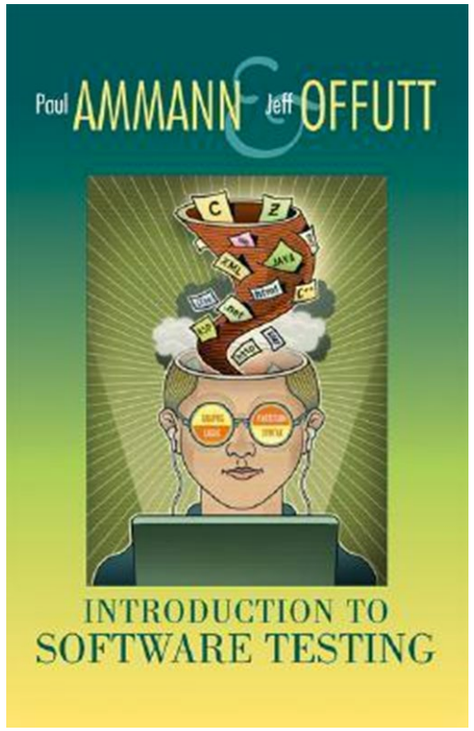
Suppose a Windows application requires users to enter their Date of Birth (in the form of DD/MM/YYYY)

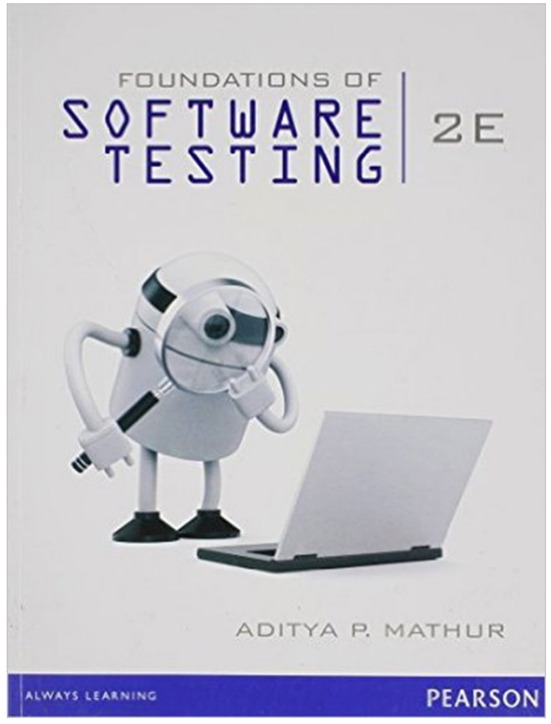
Design test cases for checking the validity of a Date of Birth.

28/03/2016
1/1/1971
10/20/1989
…

References:

- Introduction to Software Testing (1st Edition), ISBN: 978-0521880381
- Foundations of Software Testing (2nd Edition), ISBN: 978-8131794760
- K Naik and P Tripathy, Software Testing and Quality Assurance: Theory and Practice, Wiley, ISBN: 978-0-471-78911-6, 2008.





Acknowledgment:

. Dr Lingming Zhang, UT Dallas

. Dr Diana Kuo, Swinburne University of Technology

. A/Prof. Dan Hao, Peking University

Thanks!


Testing
Software
design
articles
management
web
use
test
code
quality
development
good
risk
usability
Agile
information
Project
People
article

SENG3320/6320:
Software Verification and Validation

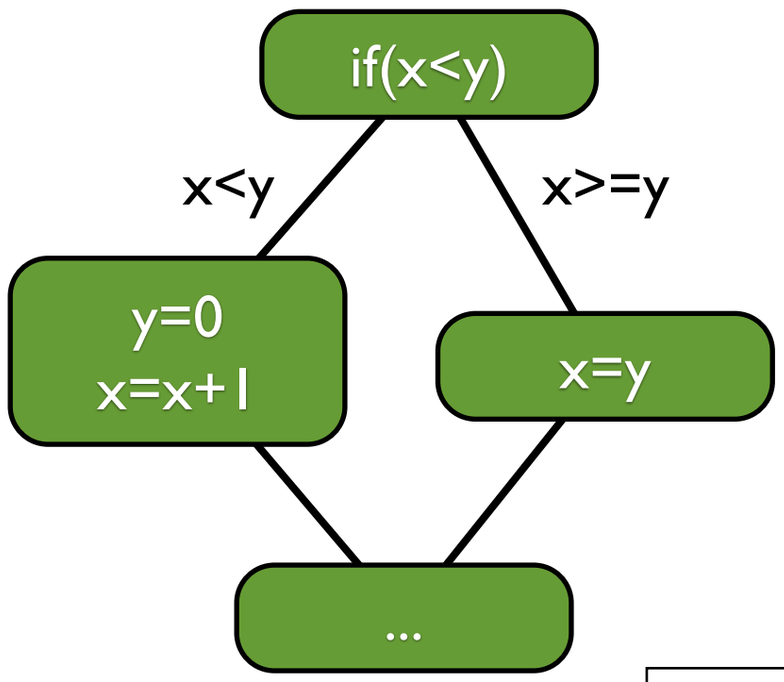
White-Box Testing

White Box testing

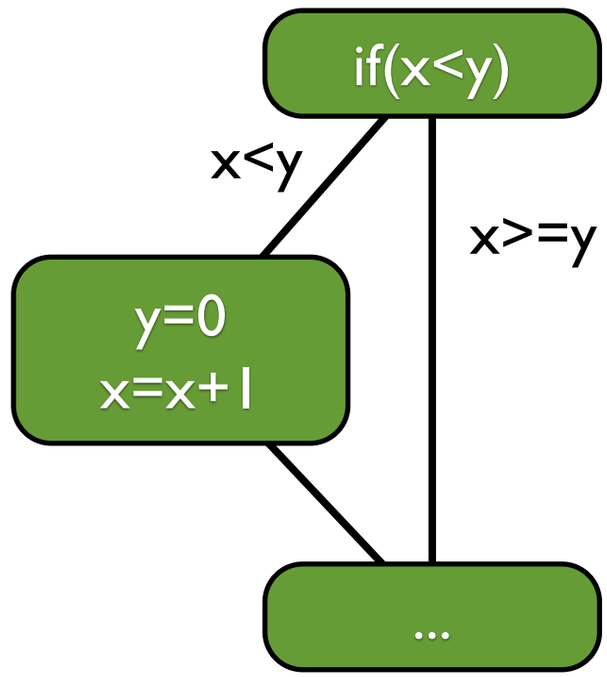
- White box testing (structural testing):
  - Generating test cases based on the structure of the program
  - A common way is to abstract program into control flow graph (CFG)
    - Node : Sequences of statements (basic block)
    - Edge : Transfers of control
- Coverage metrics:
  - ***Statement coverage***: all statements in the programs should be executed at least once
  - ***Branch coverage***: all branches in the program should be executed at least once
  - ***Path coverage***: all execution paths in the program should be executed at lest once

CFG : The if statement

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
...
```

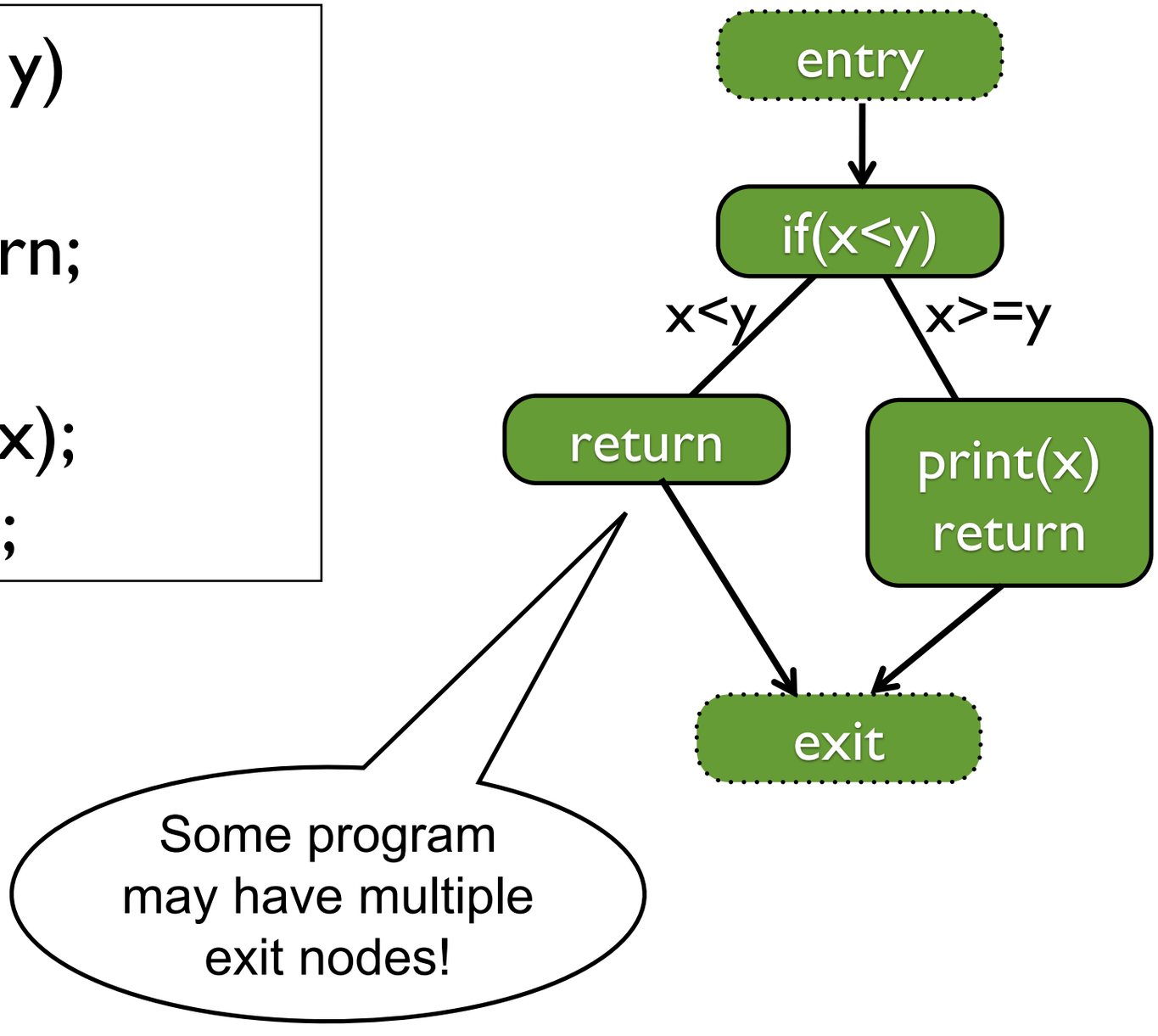



```
if (x < y)
{
   y = 0;
   x = x + 1;
}
...
```



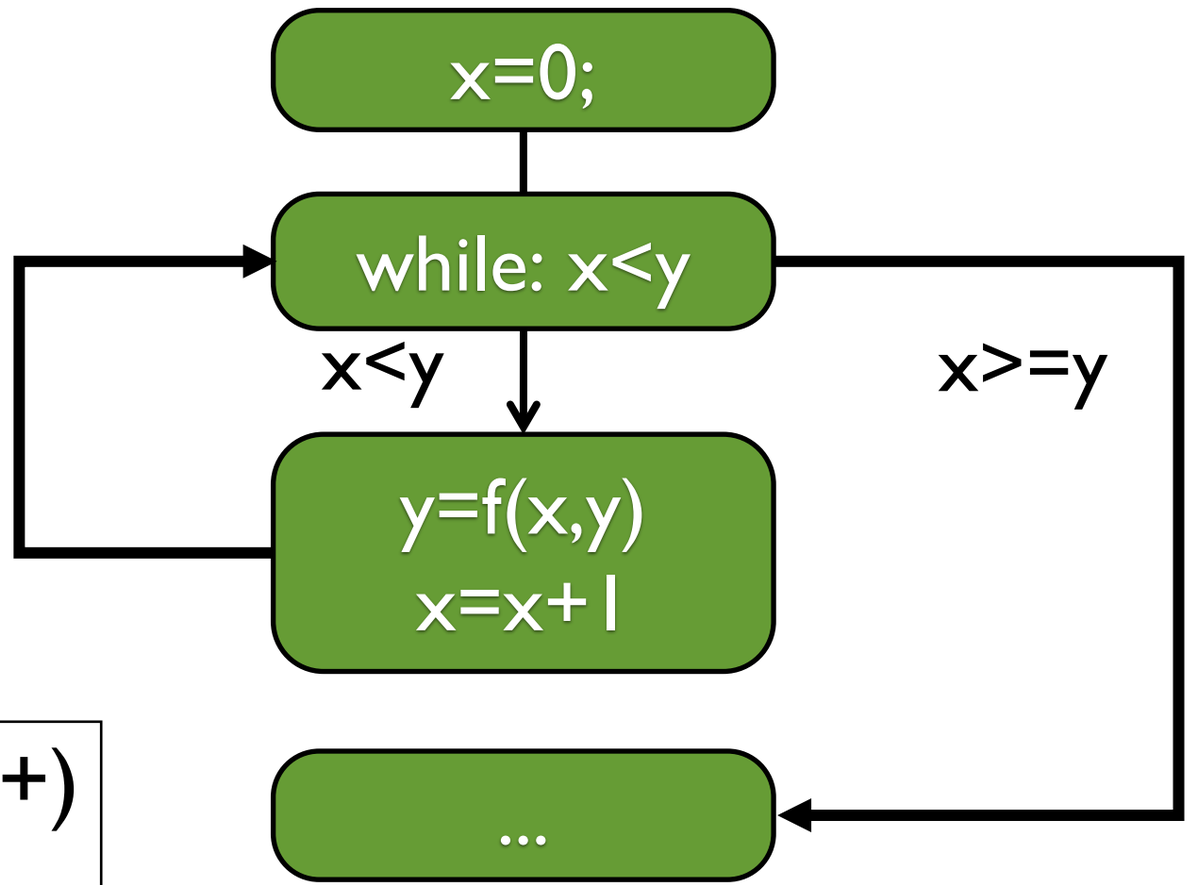
CFG : The dummy nodes

```
if (x < y)
{
   return;
}
print (x);
return;
```



CFG : while and for loops

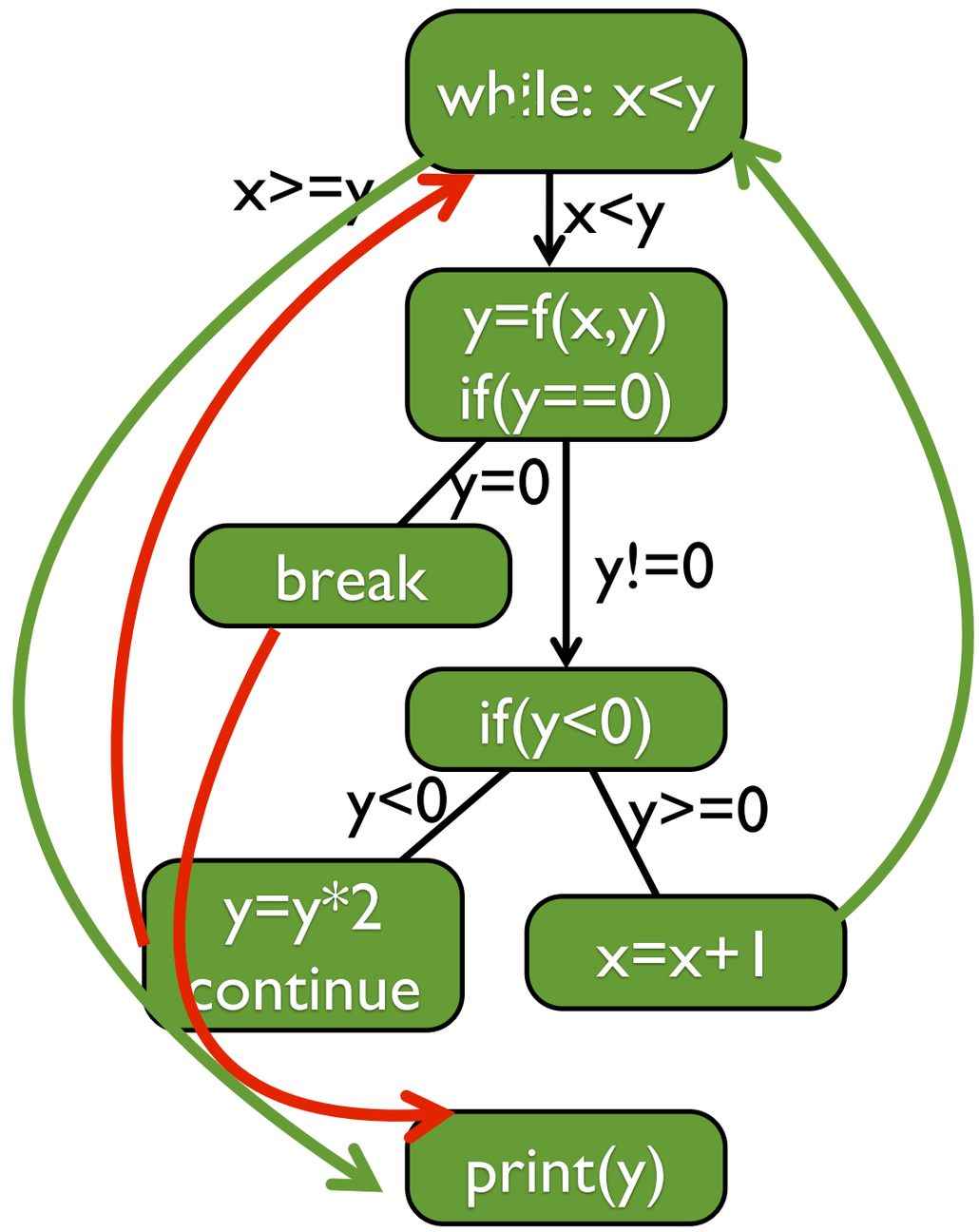
```
x=0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```




```
for (x = 0; x < y; x++)
{
   y = f (x, y);
}
```

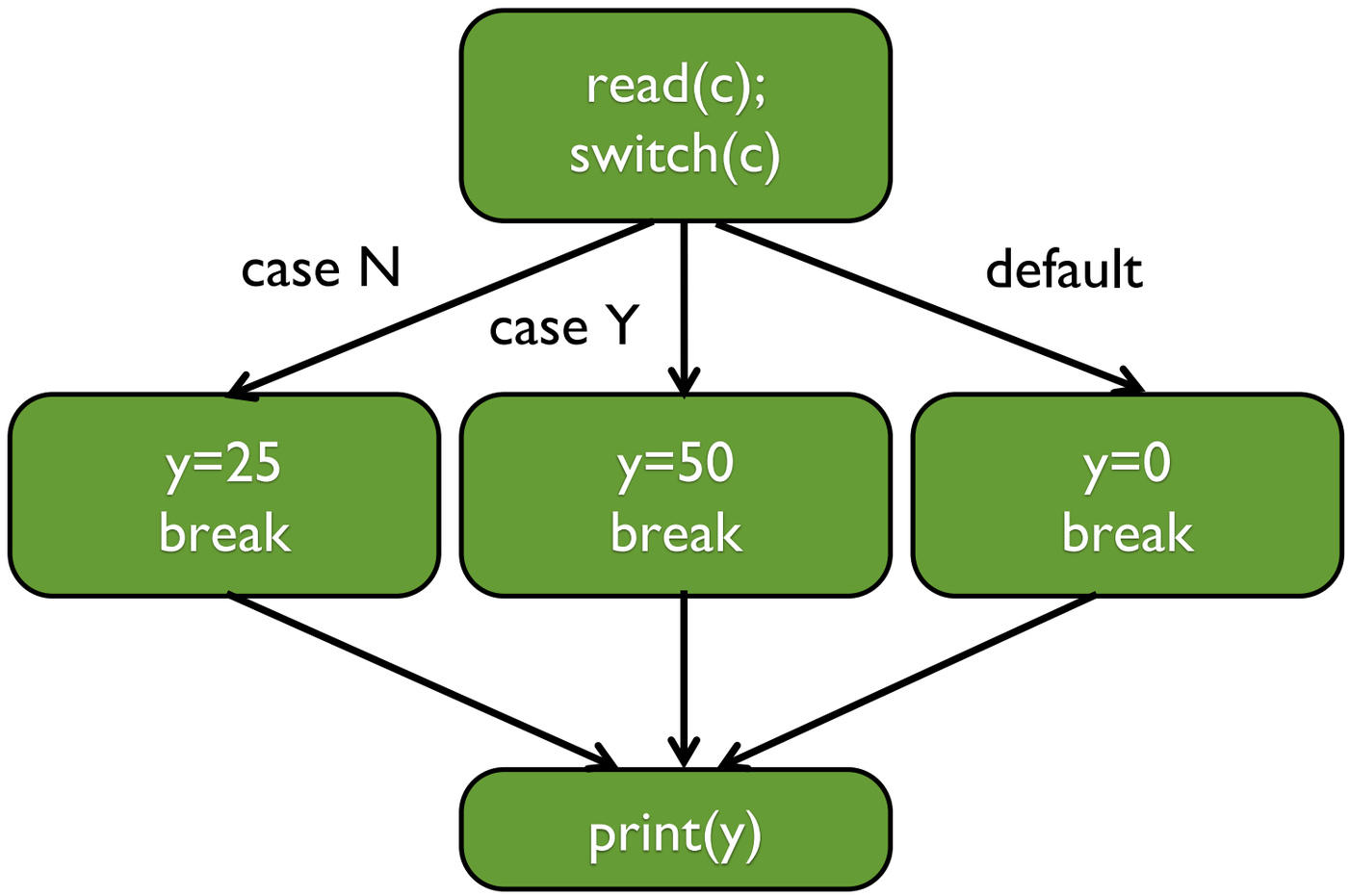
CFG: break and continue

```
while (x < y)
{
   y = f (x, y);
   if (y == 0) {
      break;
   } else if (y<0) {
      y = y*2;
      continue;
   }
   x = x + 1;
}
print (y);
```



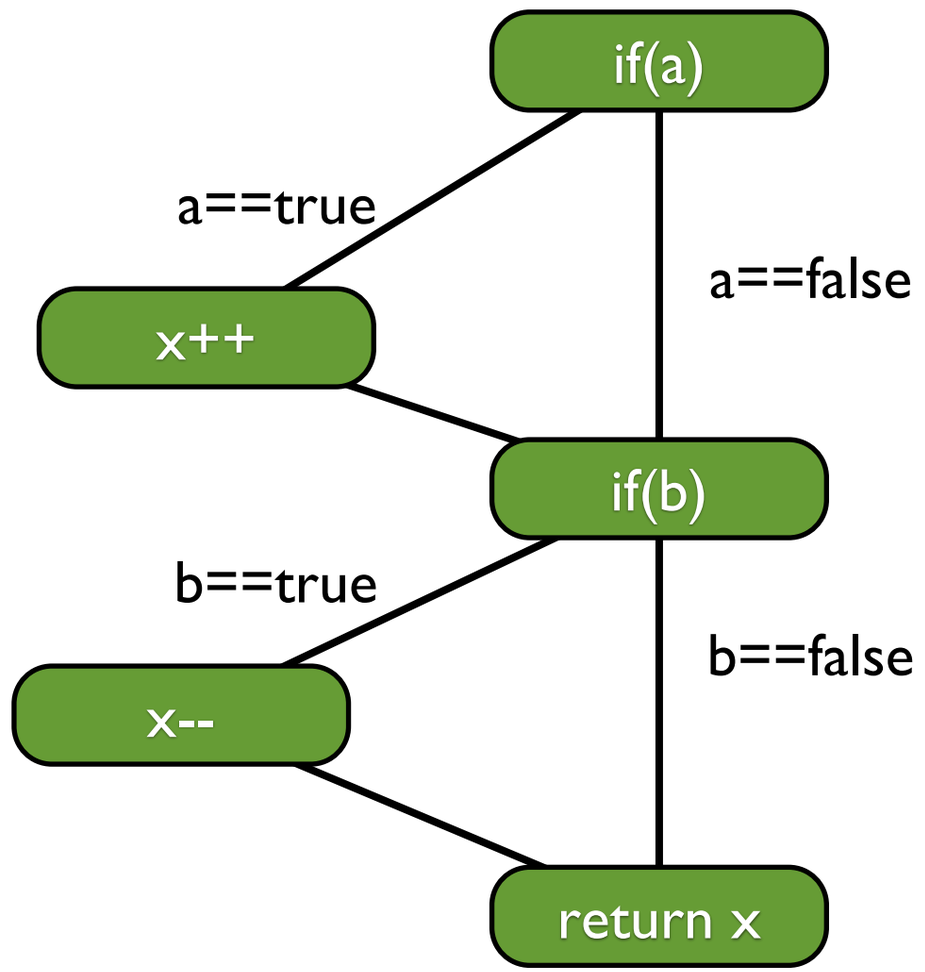
CFG: switch

```
read ( c) ;
switch ( c )
{
   case ‘N’:
      y = 25;
      break;
   case ‘Y’:
      y = 50;
      break;
   default:
      y = 0;
      break;
}
print (y);
```



CFG-based coverage: example

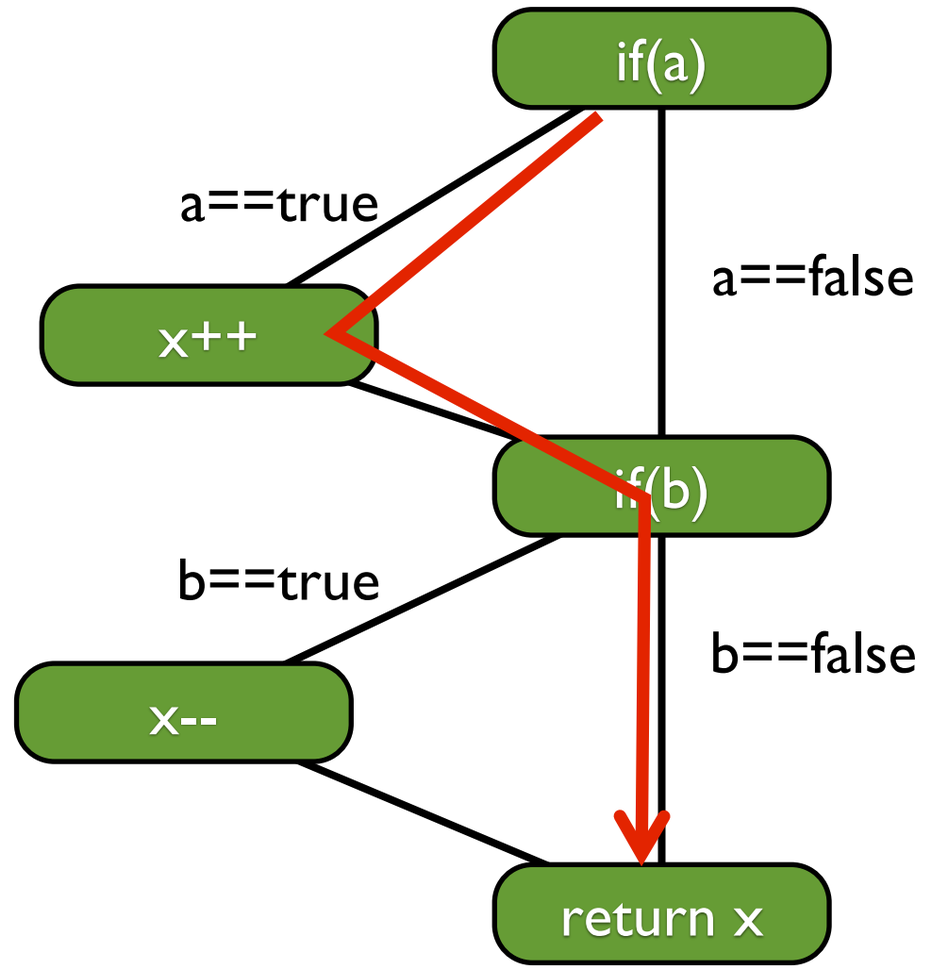
```
public class
CFGCoverageExample {
    public int testMe(int x,
boolean a, boolean b){
        if(a)
            x++;
        if(b)
            x--;
        return x;
    }
}
```



CFG-based coverage: statement coverage

- The percentage of statements covered by the test

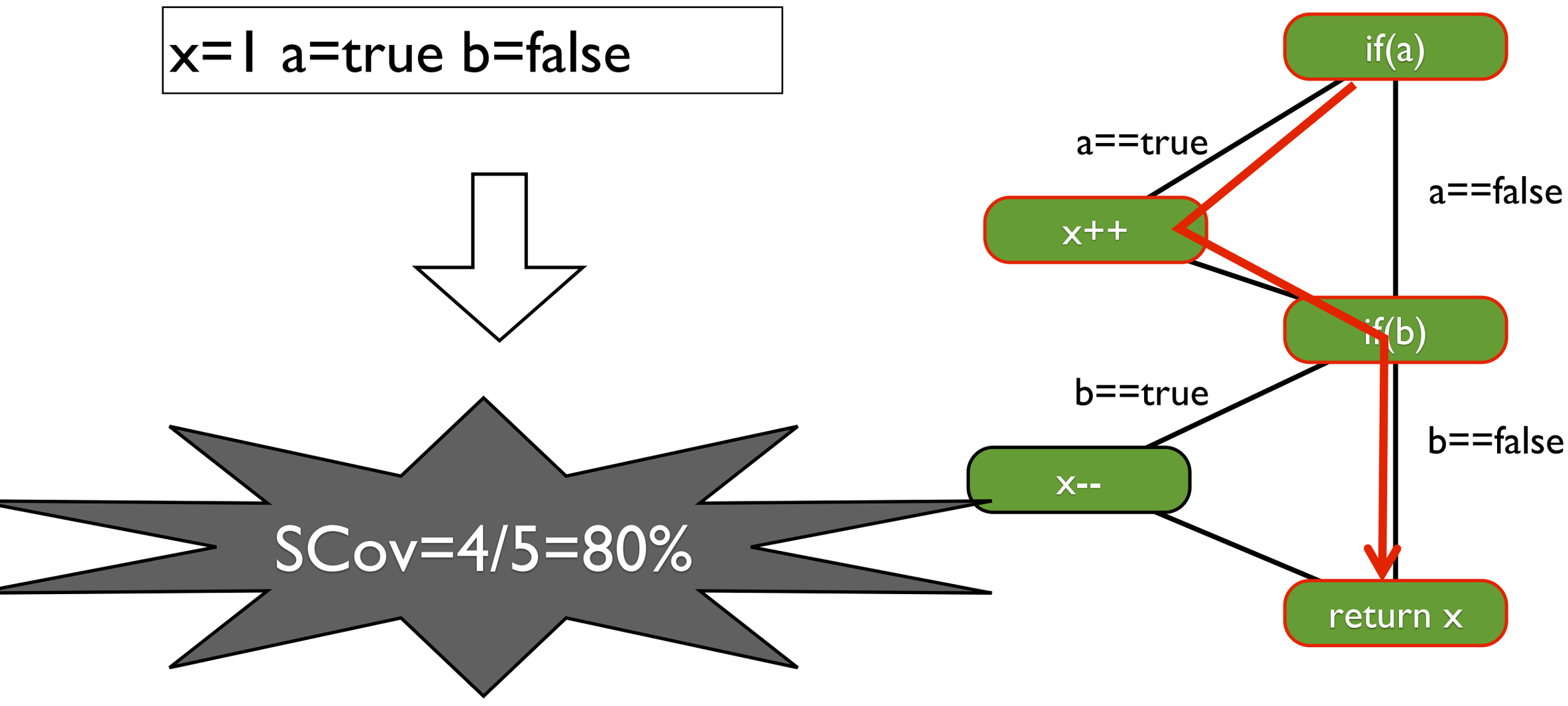
x=1 a=true b=false



CFG-based coverage: statement coverage

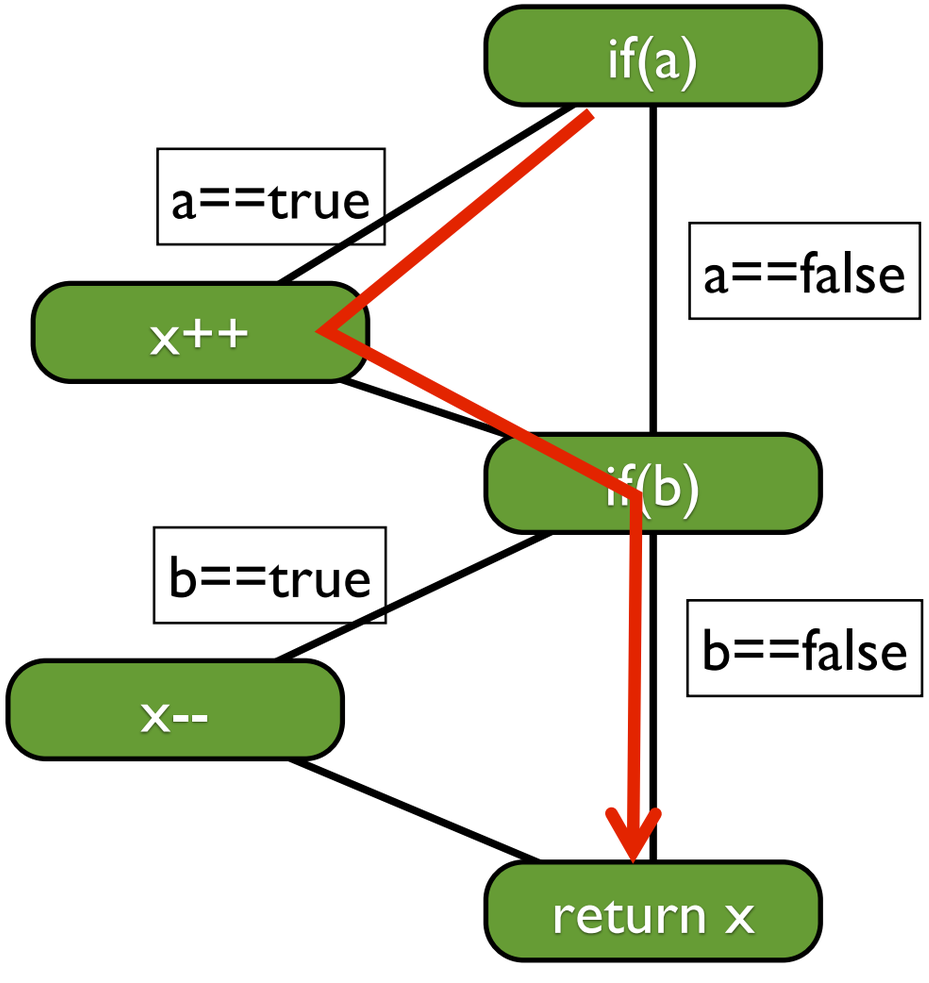
- The percentage of statements covered by the test

x=1 a=true b=false



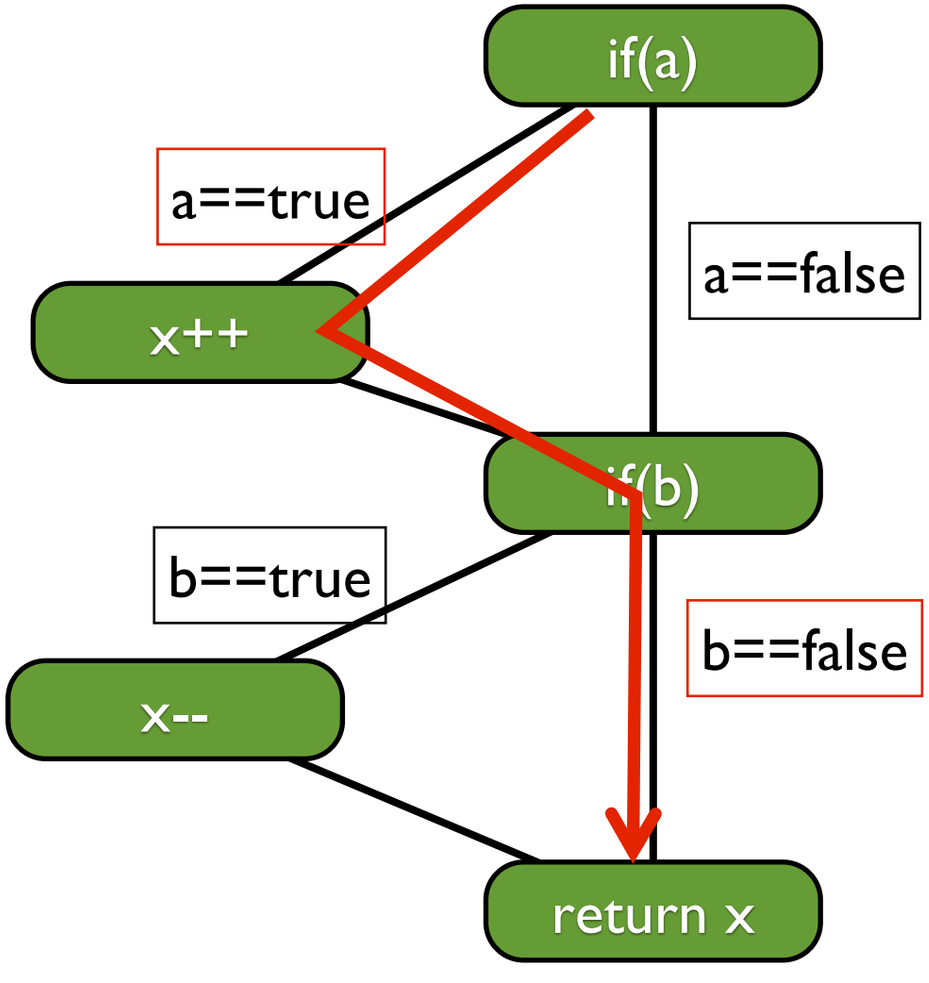
CFG-based coverage: branch coverage

- The percentage of branches covered by the test
  - Consider both false and true branch for each conditional statement



CFG-based coverage: branch coverage

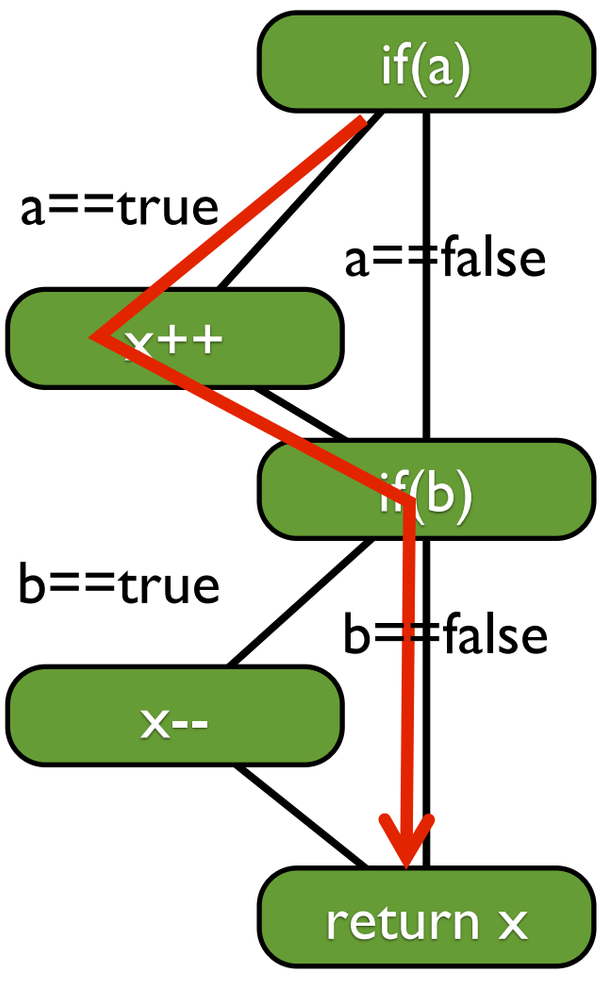
- The percentage of branches covered by the test
  - Consider both false and true branch for each conditional statement



BCov=2/4=50 %

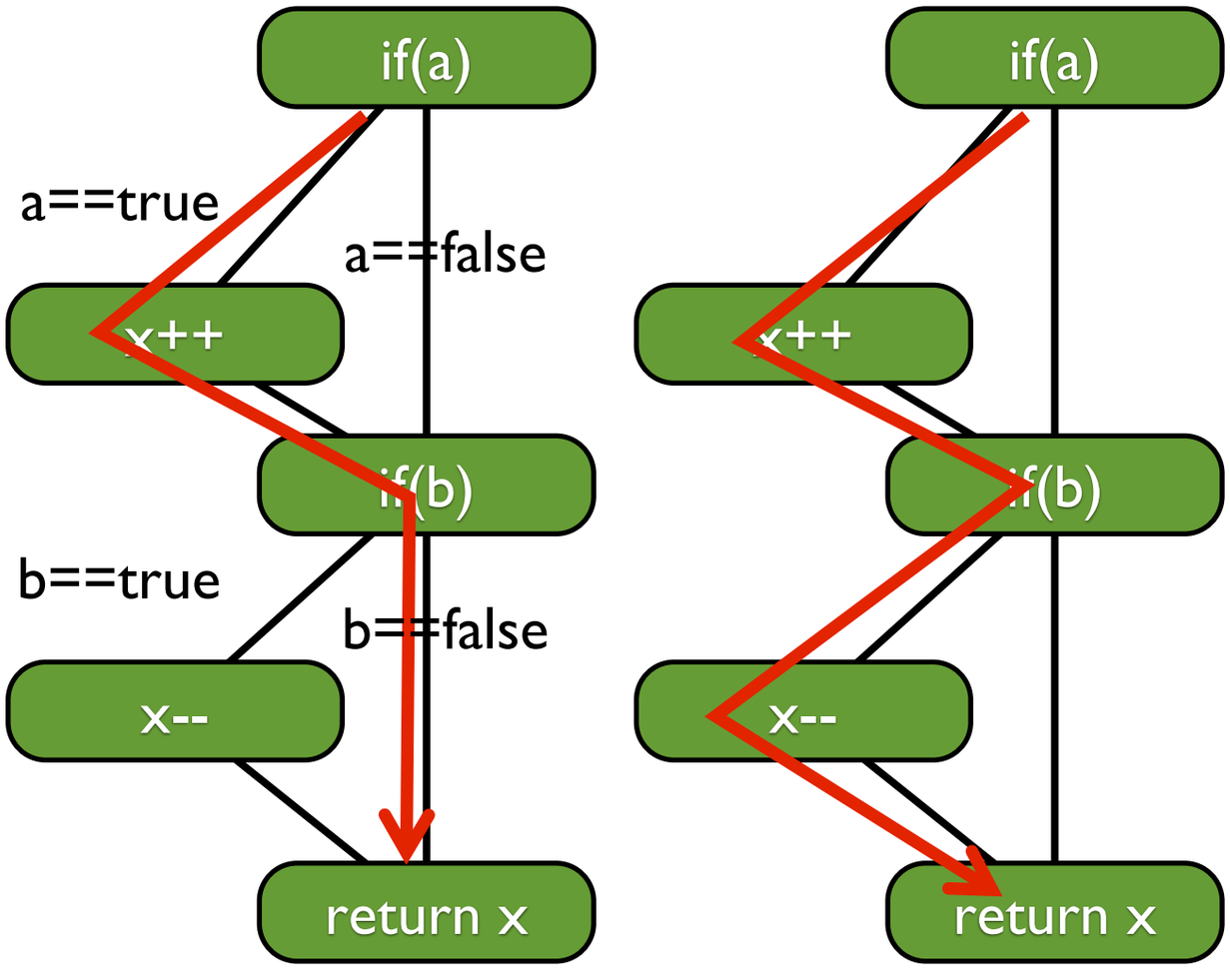
CFG-based coverage: path coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



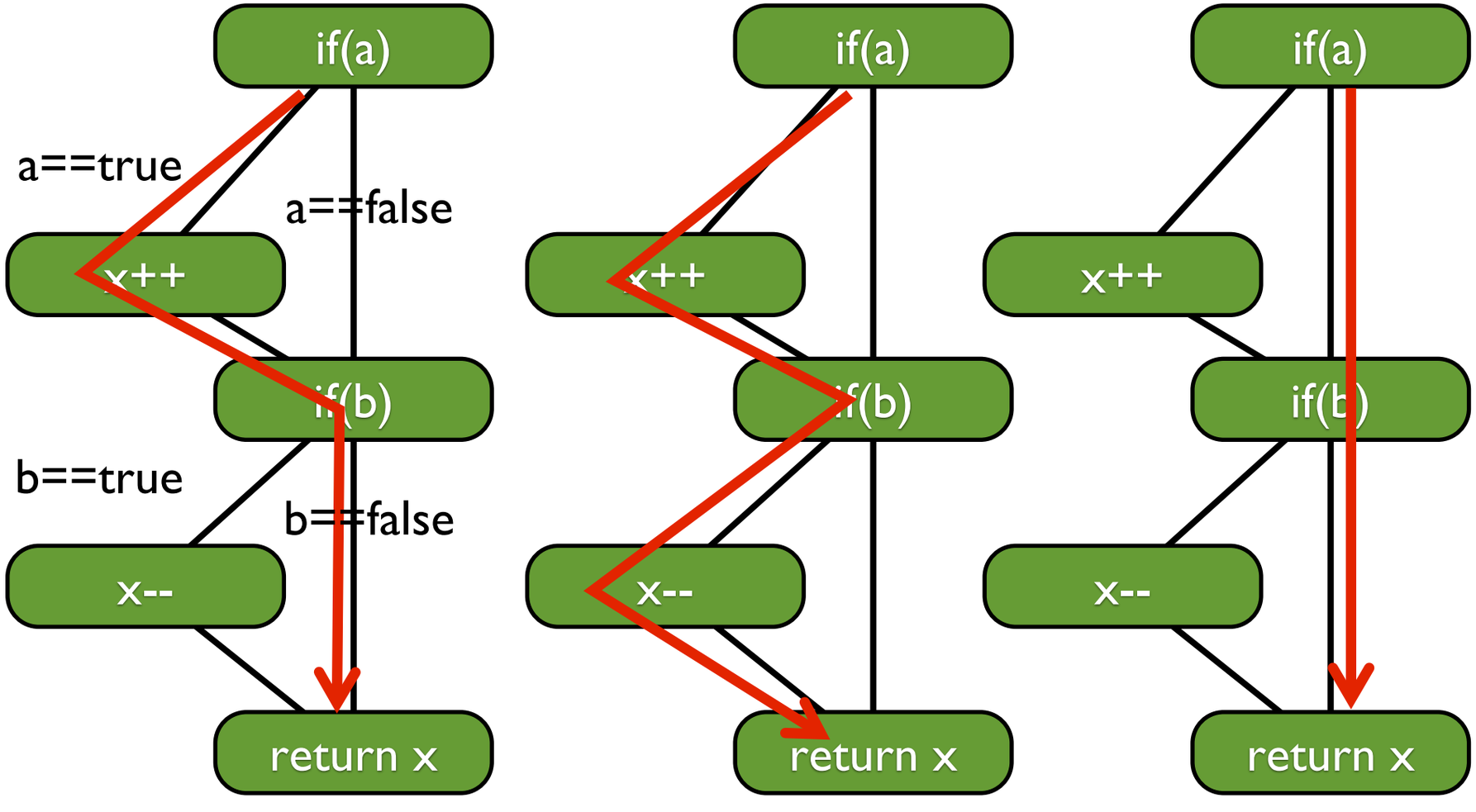
CFG-based coverage: path coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



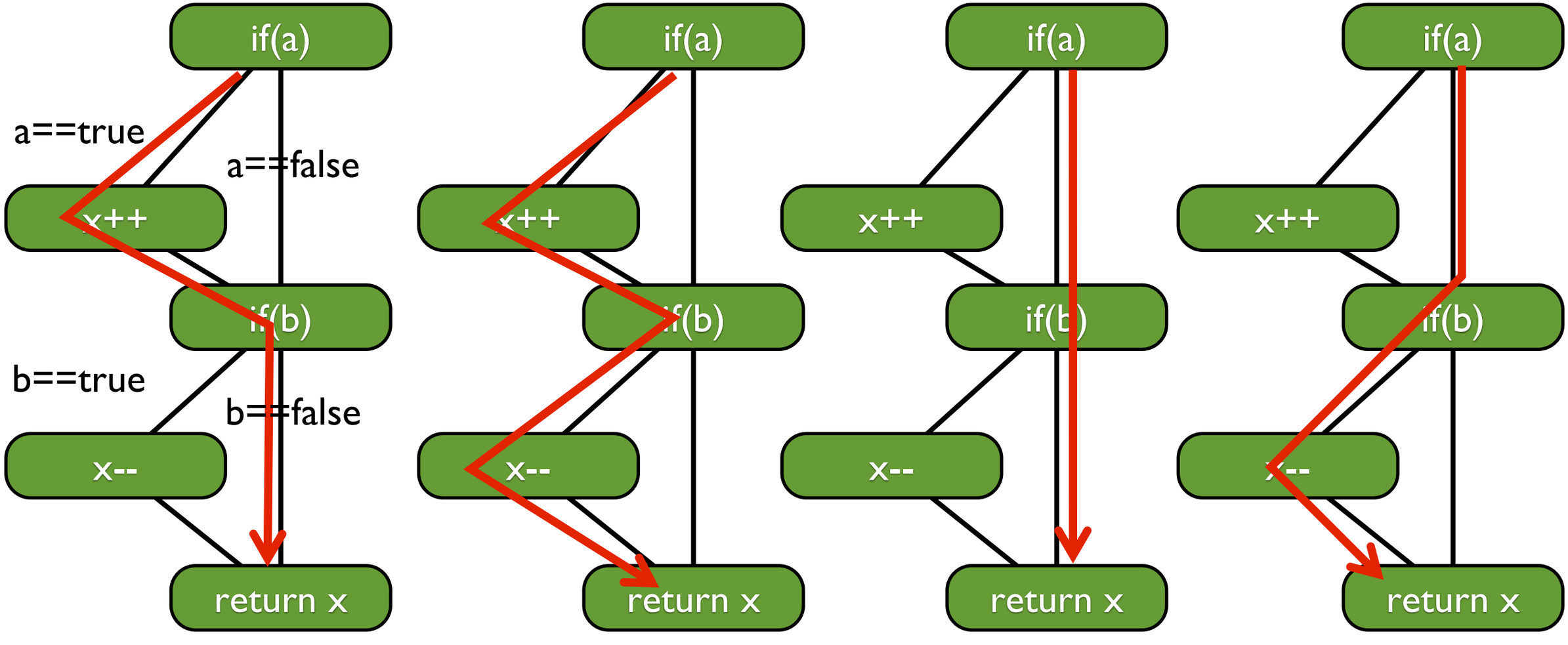
CFG-based coverage: path coverage

- The percentage of paths covered by the test
  - Consider all possible program execution paths



CFG-based coverage: path coverage

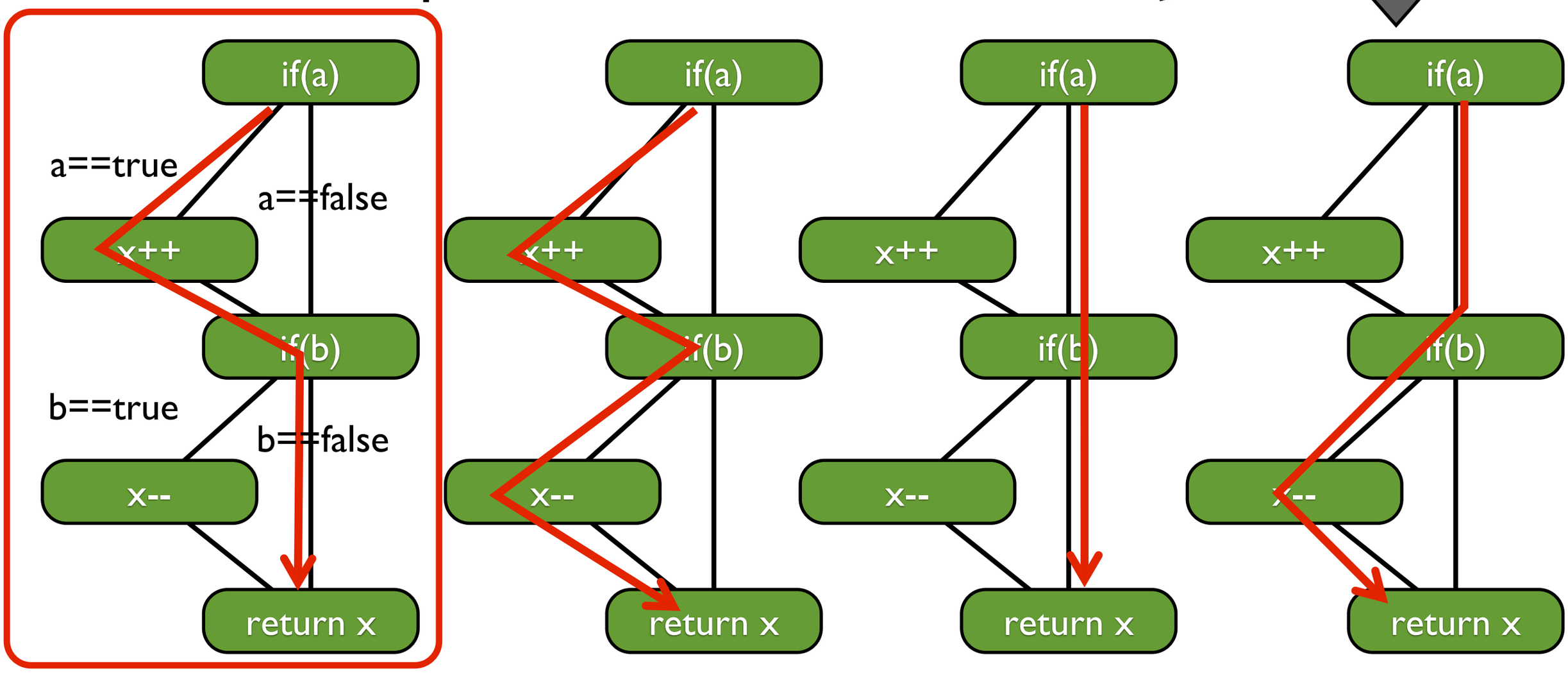
- The percentage of paths covered by the test
  - Consider all possible program execution paths



CFG-based coverage: path coverage

- The percentage of paths covered by the te
  - Consider all possible program execution paths

PCov=1/4=25%



CFG-based coverage

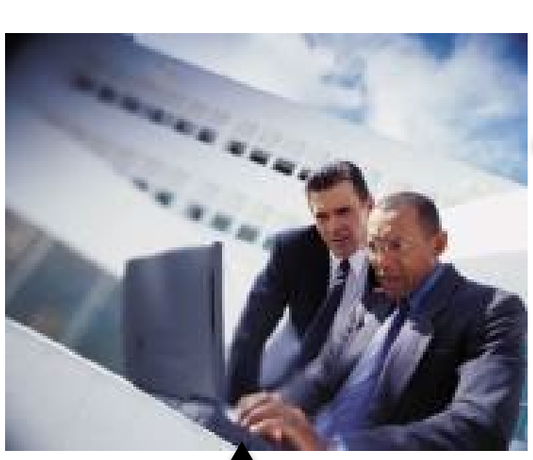
```
public class
CFGCoverageExample {
    public int testMe(int x,
boolean a, boolean b){
        if(a)
            x++;
        if(b)
            x--;
        return x;
    }
}
```

- Code coverage metrics can be used to measure test adequacy
- Usually, a test covering/executing more code may indicate better test quality

Test case: (1, true, false)

A Typical Software Testing Process

Test case generation



CFG-based coverage: comparison

```
public class CFGCoverageExample {
        public int testMe(int x, boolean a, boolean
b){
                if(a)
                        x++;
                if(b)
                        x--;
                return x;
        }
}
```

Test case: (0, true, false)

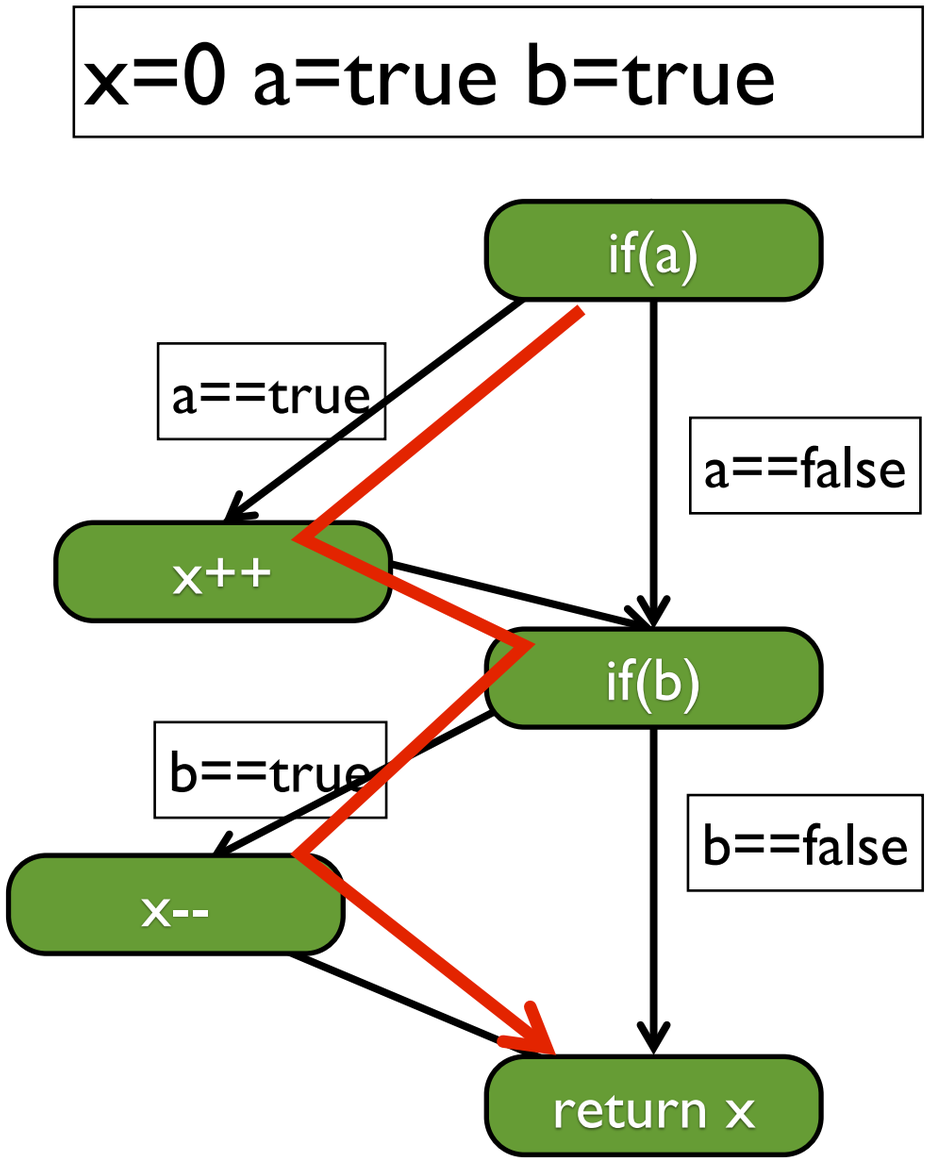
Statement coverage: 80%
Branch coverage: 50%
Path coverage: 25%

If we achieve 100% branch coverage, do we get 100% statement coverage for free?

If we achieve 100% path coverage, do we get 100% branch coverage for free?

Statement coverage VS. branch coverage

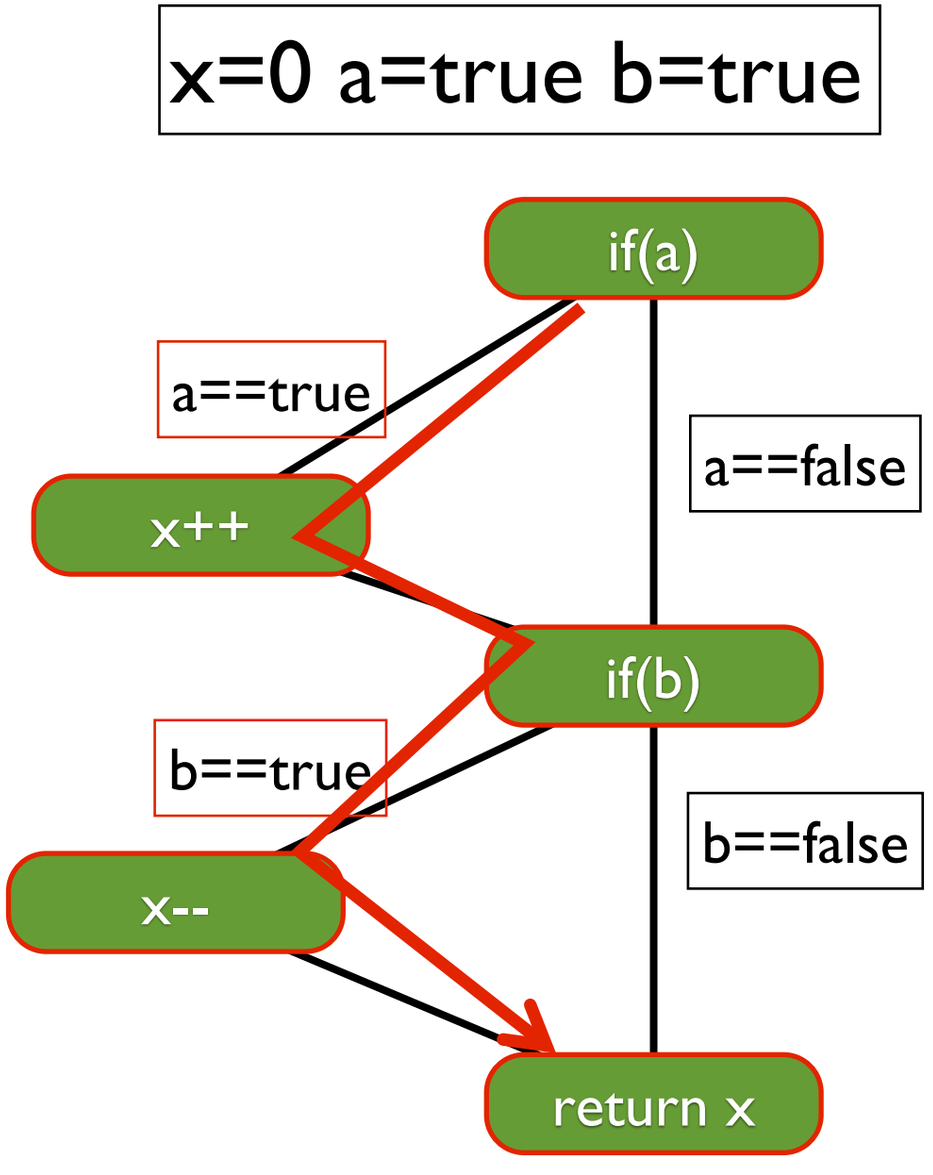
- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
  - The statements not in branches will be covered by any test
  - All other statements are in certain branch
- If a test suite achieve 100% s-coverage, will it achieve 100% b-coverage?



Statement coverage VS. branch coverage

- If a test suite achieve 100% b-coverage, it must achieve 100% s-coverage
  - The statements not in branches will be covered by any test
  - All other statements are in certain branch

Branch coverage strictly subsumes statement coverage

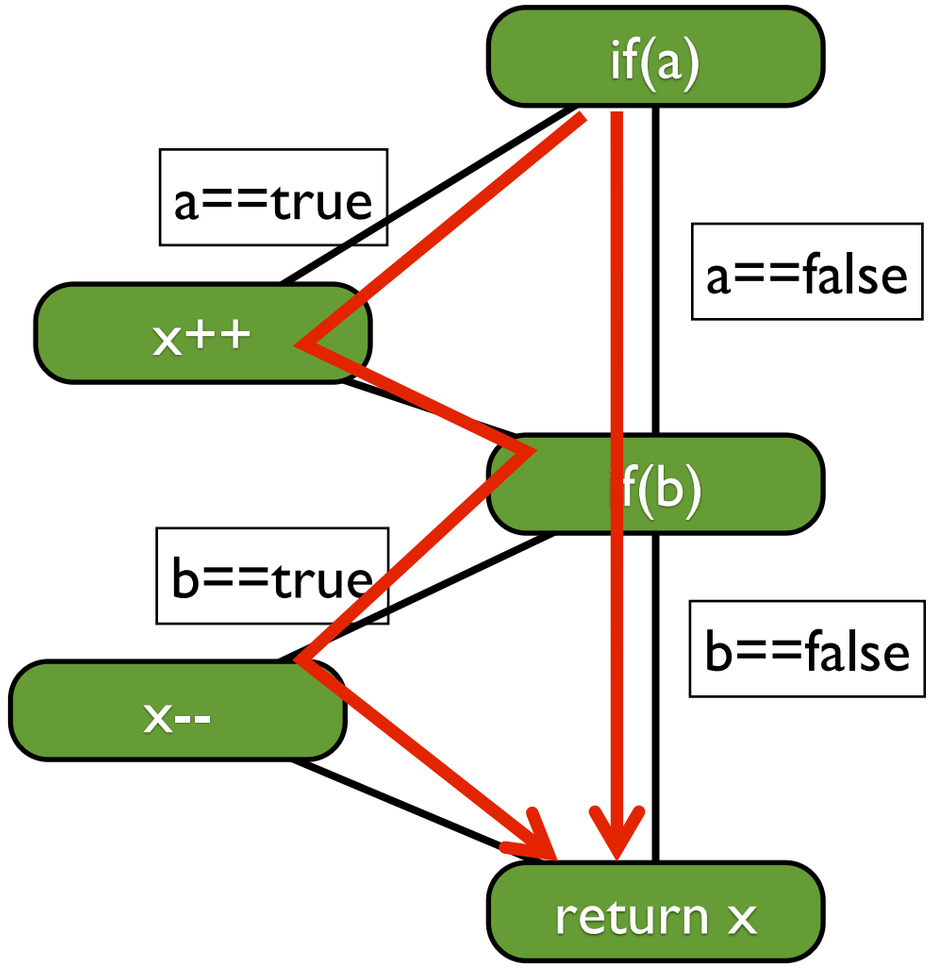


Branch coverage VS. path coverage

- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
  - All the branch combinations have been covered indicate all branches are covered
- If a test suite achieve 100% b-coverage, will it achieve 100% p-coverage?

x=0 a=true b=true

x=0 a=false b=false

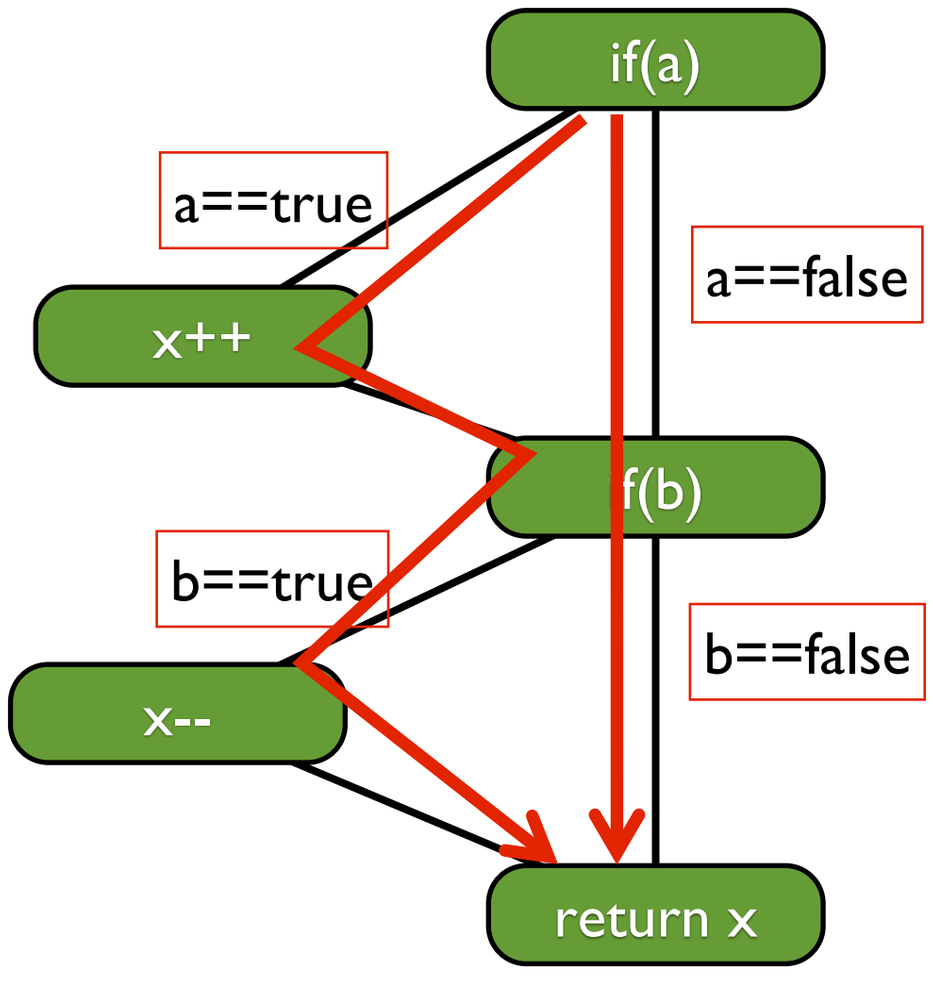


Branch coverage VS. path coverage

- If a test suite achieve 100% p-coverage, it must achieve 100% b-coverage
  - All the branch combinations have been covered indicate all branches are covered

x=0 a=true b=true

x=0 a=false b=false



Path coverage strictly subsumes branch coverage

CFG-based coverage: comparison summary

Path coverage
strictly subsumes branch coverage
strictly subsumes statement coverage

Path Coverage

↓

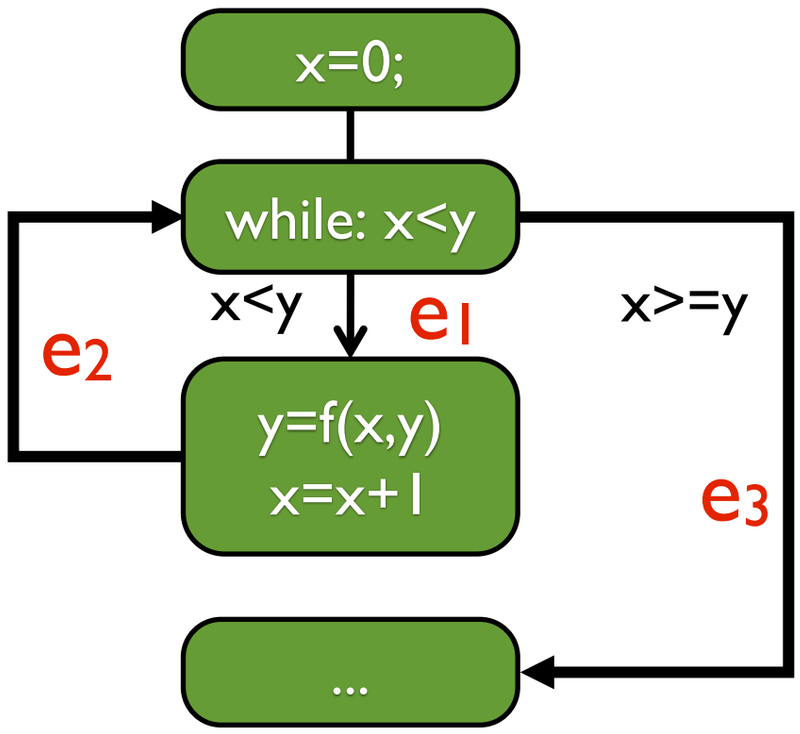
Branch Coverage

↓

Statement Coverage

Should we just use path coverage?

```
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
...
```



Possible Paths

e3

e1e2e3

e1e2e1e2e3

e1e2e1e2e1e2e3

...

Path coverage can be infeasible for real-world programs

CFG-based coverage: limitation

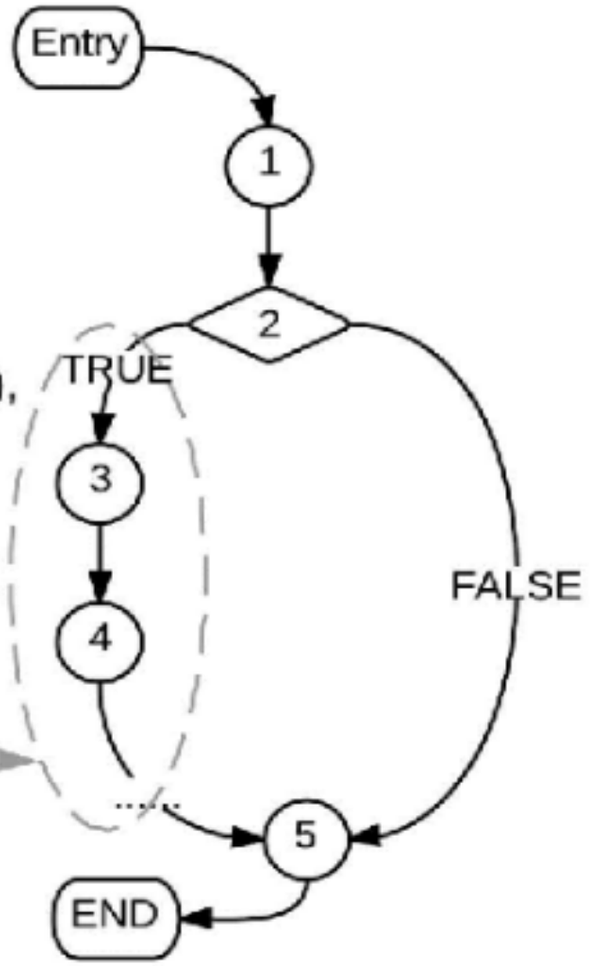
100% coverage may not be possible due to infeasible paths/branches.

```
1  ......
2  if ((cacheLen > 0) && (Y != NULL) {
3    count += (cacheLen + counter.m_countNodesStartCount);
4    if (cacheLen > 0)
5      appendBtoFList(counter.m_countNodes,m_newFound);
6    m_newFound.removeAllElements();
7    return count;
   }
8  ......
```

```
void foo (int begin, int textLenth) {
1.  ttype = Token.SKIP;

2.  if (createToken && token
            ==null && ttype!=Token.SKIP ) {
3.      token = makeToken(ttype);
4.      token.setText(new String(text.getBuffer(),
            begin, textLenth));
        ........
    }

5.      returnToken = token;
}
```

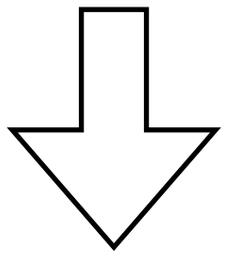


Infeasible Branch

CFG-based coverage: limitation

- 100% coverage of some aspect is never a guarantee of bug-free software

Test case: (1, 0)



```
public int sum(int x, int y){
        return x-y; //should be x+y
    }
```

Statement coverage: 100%

Branch coverage: 100%

Path coverage: 100%

Failed to detect the bug...

Software Testing Tools – JUnit

```
package test;

public class myClass {

    public static void main(String[] args) {
       myClass t = new myClass();
        int n = t.checkbool(10, true, false);
       System.out.println("the check boolean result is " + n);

         n = t.checkbool(11, false, false);
       System.out.println("the check boolean result is " + n);

          n = t.checkbool(10, true, true);
       System.out.println("the check boolean result is " + n);
}
   public int checkbool(int x, boolean a, boolean b) {
        if(a)
              x++;
        if(b)
                    x--;
        return x;
    }
}
```

```
the check boolean result is 11
the check boolean result is 11
the check boolean result is 10
```

Tool support - testing framework

- xUnit
  - Created by Kent Beck in 1989
    - This is the same guy who invented XP and TDD
    - The first one was sUnit (for smalltalk)
    - There are about 70 xUnit frameworks for corresponding languages
- JUnit
  - A simple, flexible, easy-to-use, open-source, and practical xUnit framework for Java.
  - Can deal with a large and extensive set of test cases.
  - Refer to www.junit.org.

Terms

- Test case: a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement
- Test oracle: The expected outputs of software for given input. It is a part of a test case.
- Test driver: a software framework that can load a collection of test cases or a test suite.
- Test suite: a collection of test cases.

```java
package test;


import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class junit_test {
myClass t;


@Test
void testBool_1() {
t = new myClass();
assertEquals(11, t.checkbool(10, true, false));
assertEquals(10, t.checkbool(10, true, true));
}

}
```

Test case

Test oracle

assertEquals([msg], expected, actual)	Check that the two values are equal

Limitation: Not clear which test case caused the failure

```
package test;


import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class junit_test {
myClass t;


@Test
void testBool_1() {
t = new myClass();
assertEquals(11, t.checkbool(10, true, false));
}

@Test
void testBool_2() {
t = new myClass();
assertEquals(10, t.checkbool(10, true, true));
}

}
```

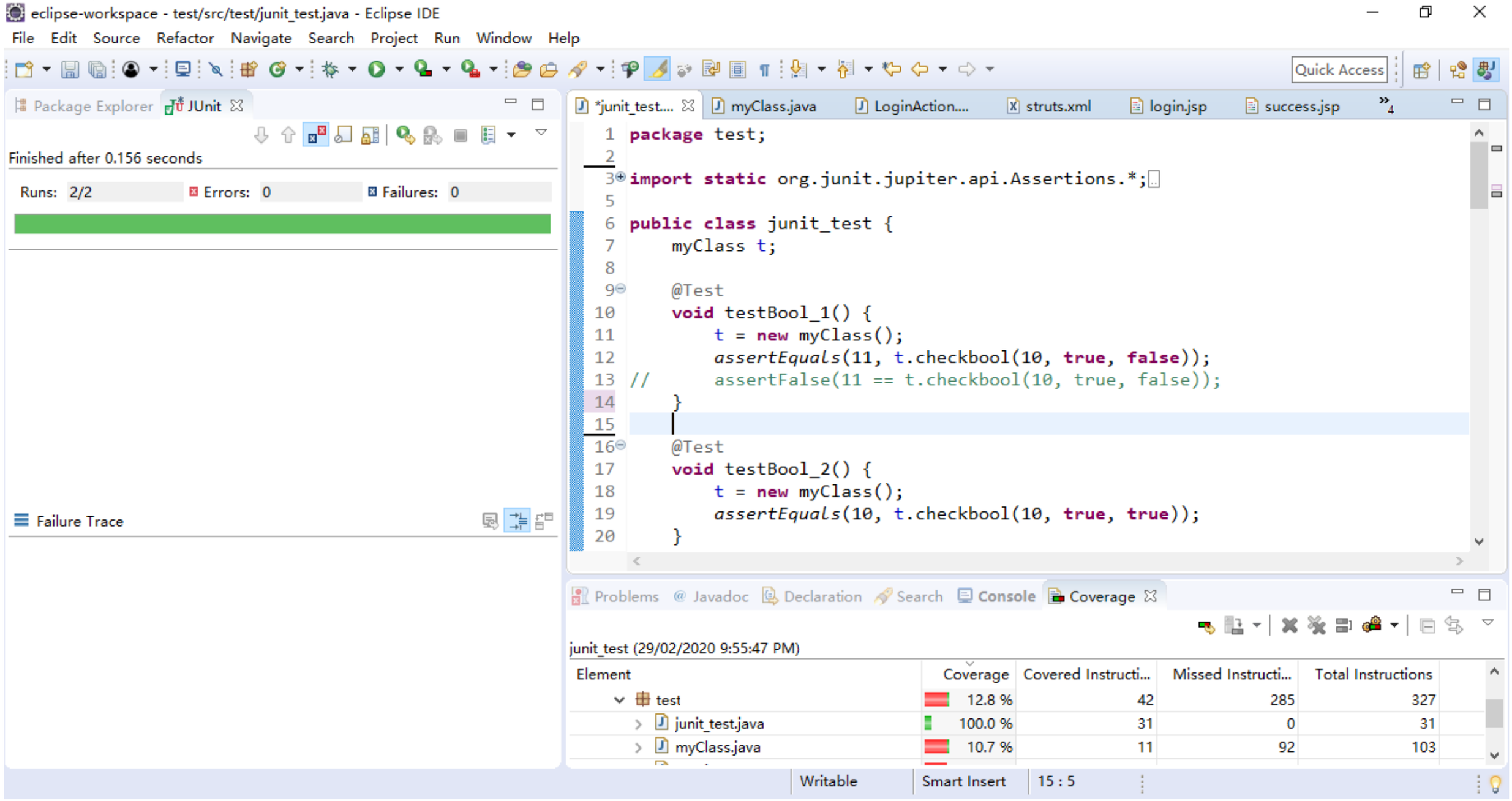
Test case

Test oracle

A Better Way

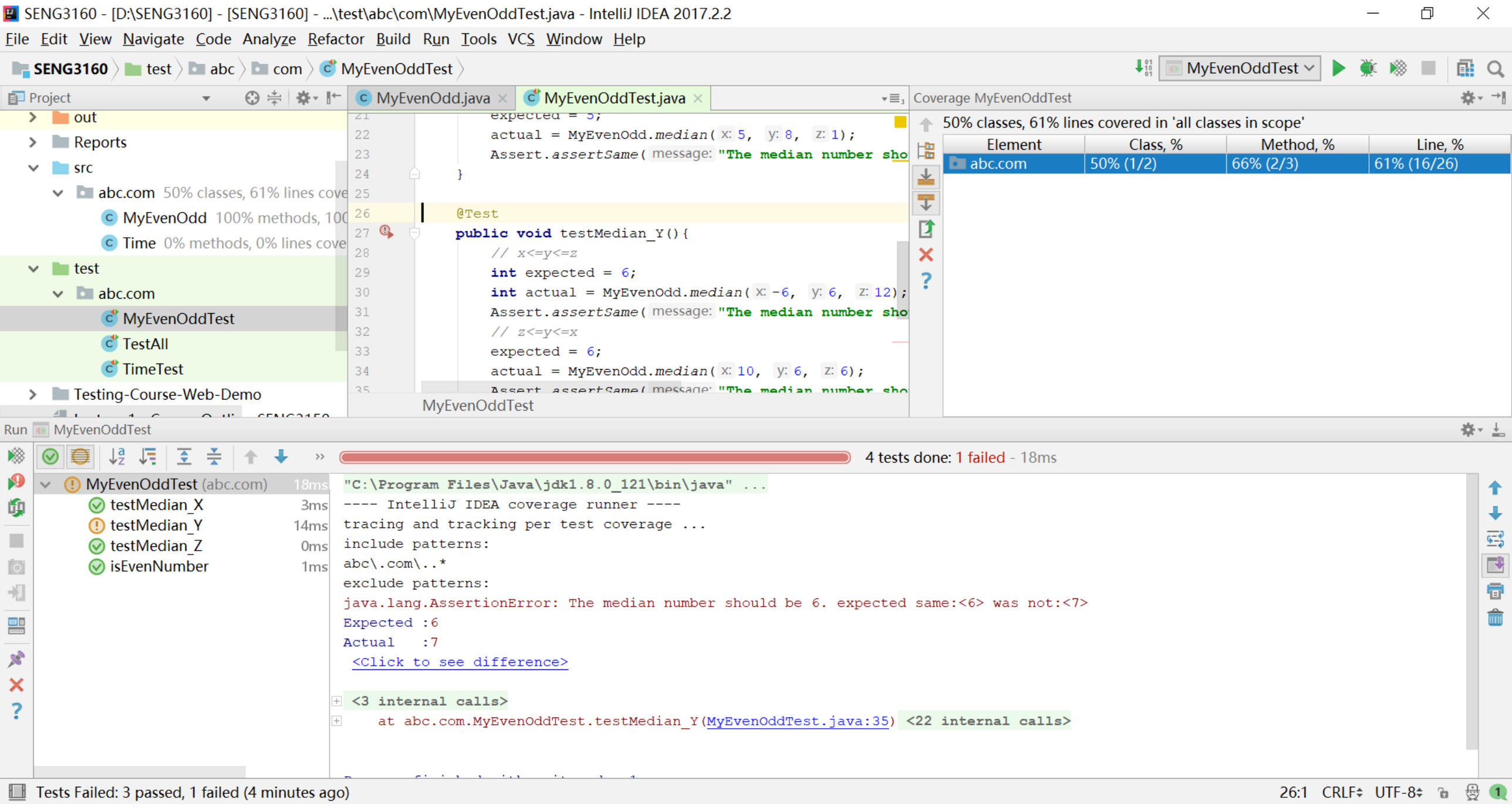
Eclipse Support

- Junit is supported by Eclipse



IntelliJ Support

- Junit is also supported by others IDEs such as IntelliJ



[Prepare for testing—IntelliJ IDEA (jetbrains.com)]

JUnit: annotations

Annotation	Description
@Test	Identify test methods
@Test (timeout=100)	Fail if the test takes more than 100ms
@Before	Execute before each test method
@After	Execute after each test method
@BeforeClass	Execute before each test class
@AfterClass	Execute after each test class
@Ignore	Ignore the test method

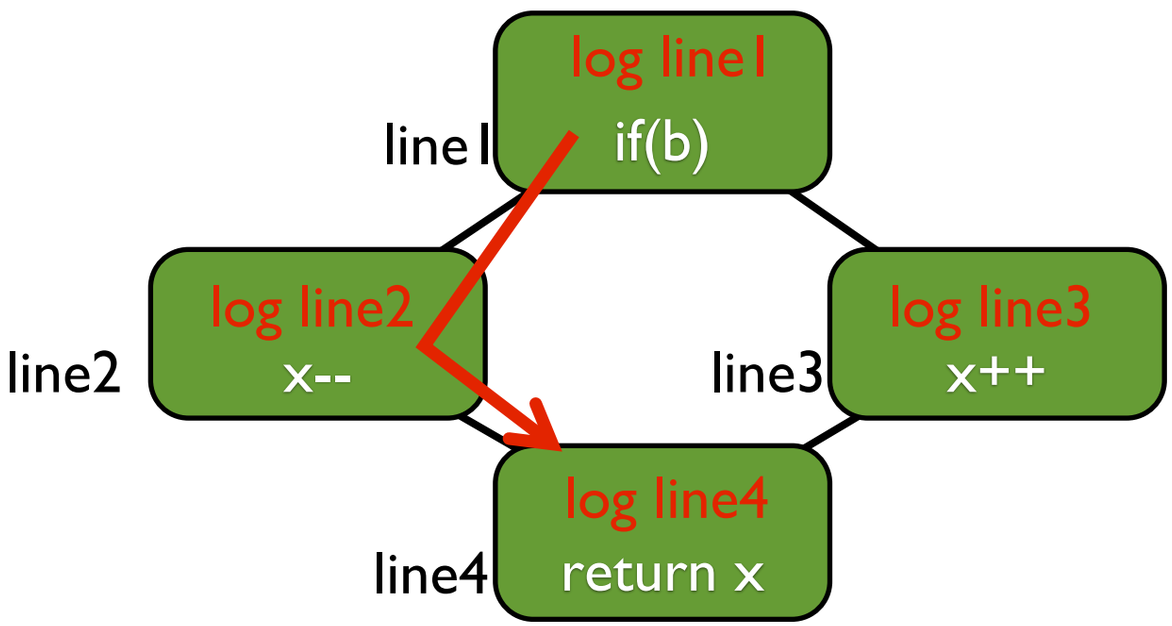
JUnit: assertions

Assertion	Description
fail([msg])	Let the test method fail, optional msg
assertTrue([msg], bool)	Check that the boolean condition is true
assertFalse([msg], bool)	Check that the boolean condition is false
assertEquals([msg], expected, actual)	Check that the two values are equal
assertNull([msg], obj)	Check that the object is null
assertNotNull([msg], obj)	Check that the object is not null
assertSame([msg], expected, actual)	Check that both variables refer to the same object
assertNotSame([msg], expected, actual)	Check that variables refer to different objects

Software Testing Tools – EclEmma

Coverage collection: mechanism

- The code under test is instrumented (source/binary)
  - Log code that writes to a trace file is inserted in every branch, statement etc.
- When the instrumented code is executed, the coverage info will be written to trace file



Coverage file

line1
line2
line4

Tool support - coverage collection

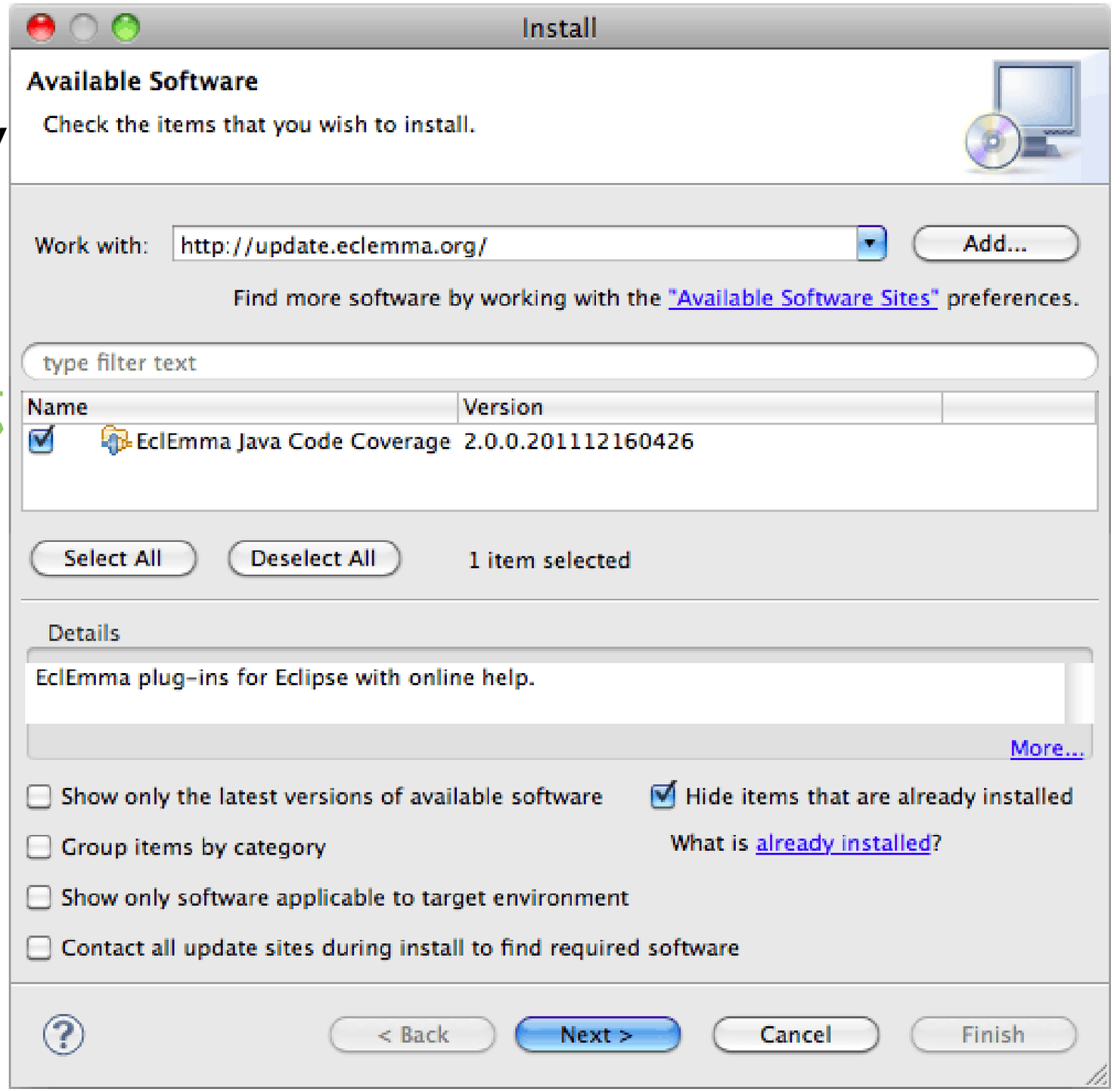
- Emma: http://emma.sourceforge.net/
- EclEmma: http://www.eclemma.org/installation.html/
- Cobertura: http://cobertura.github.io/cobertura/
- Clover: https://www.atlassian.com/software/clover/overview
- JCov: https://wiki.openjdk.java.net/display/CodeTools/jcov
- JaCoCo: http://www.eclemma.org/jacoco/

EclEmma: installation

- From your Eclipse menu select Help → Install New Software...
- In the Install dialog enter http://update.eclemma.org/ at the Work with field
- Check the latest EclEmma version and press Next
- Follow the steps in the installation wizard.

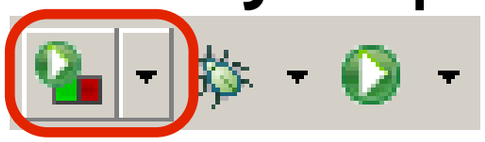
EclEmma: installation

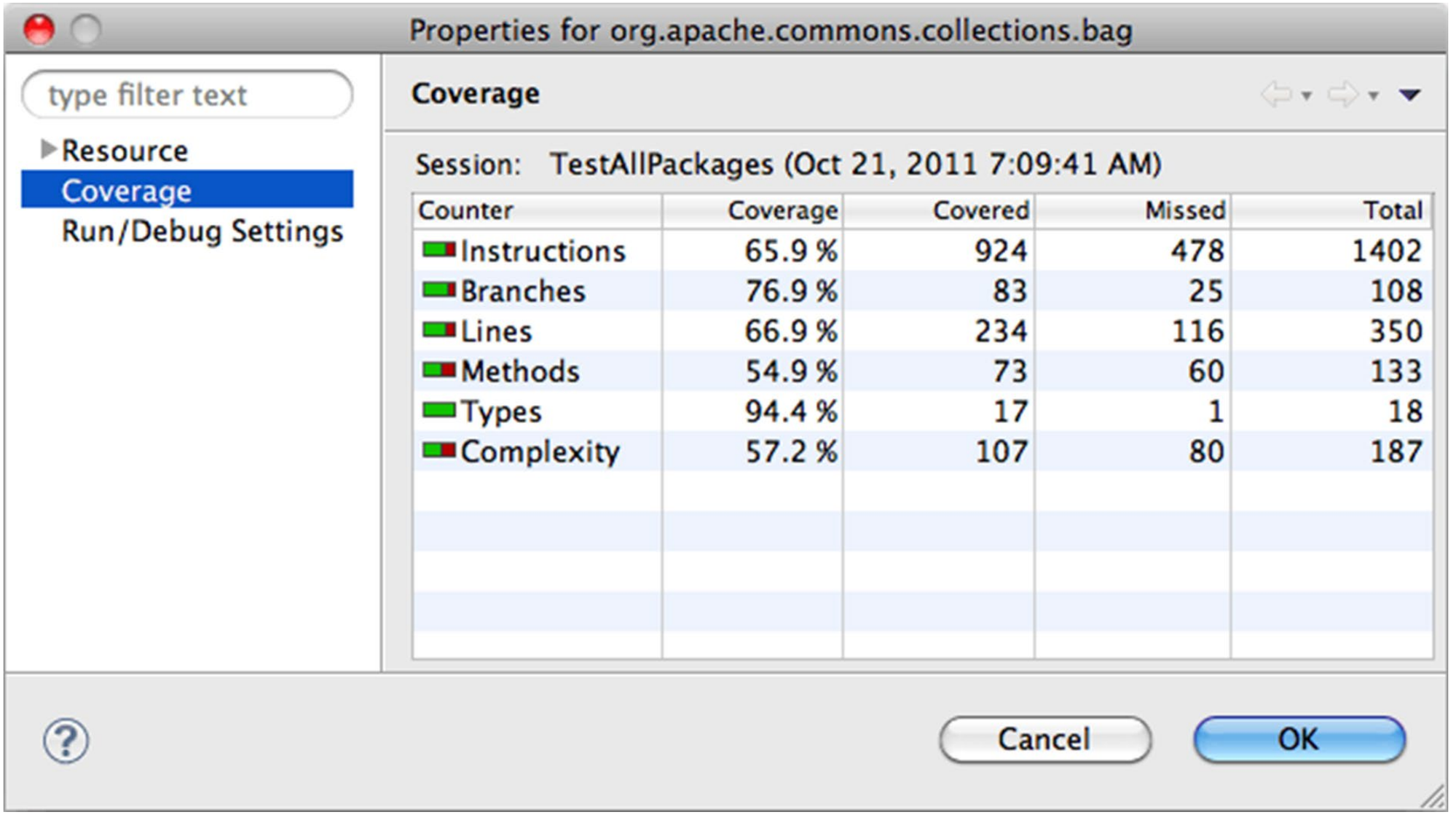
- From your Eclipse menu select Help → Install New Software...
- In the Install dialog enter http://update.eclemma.org/ at the Work with field
- Check the latest EclEmma version and press Next
- Follow the steps in the installation wizard.



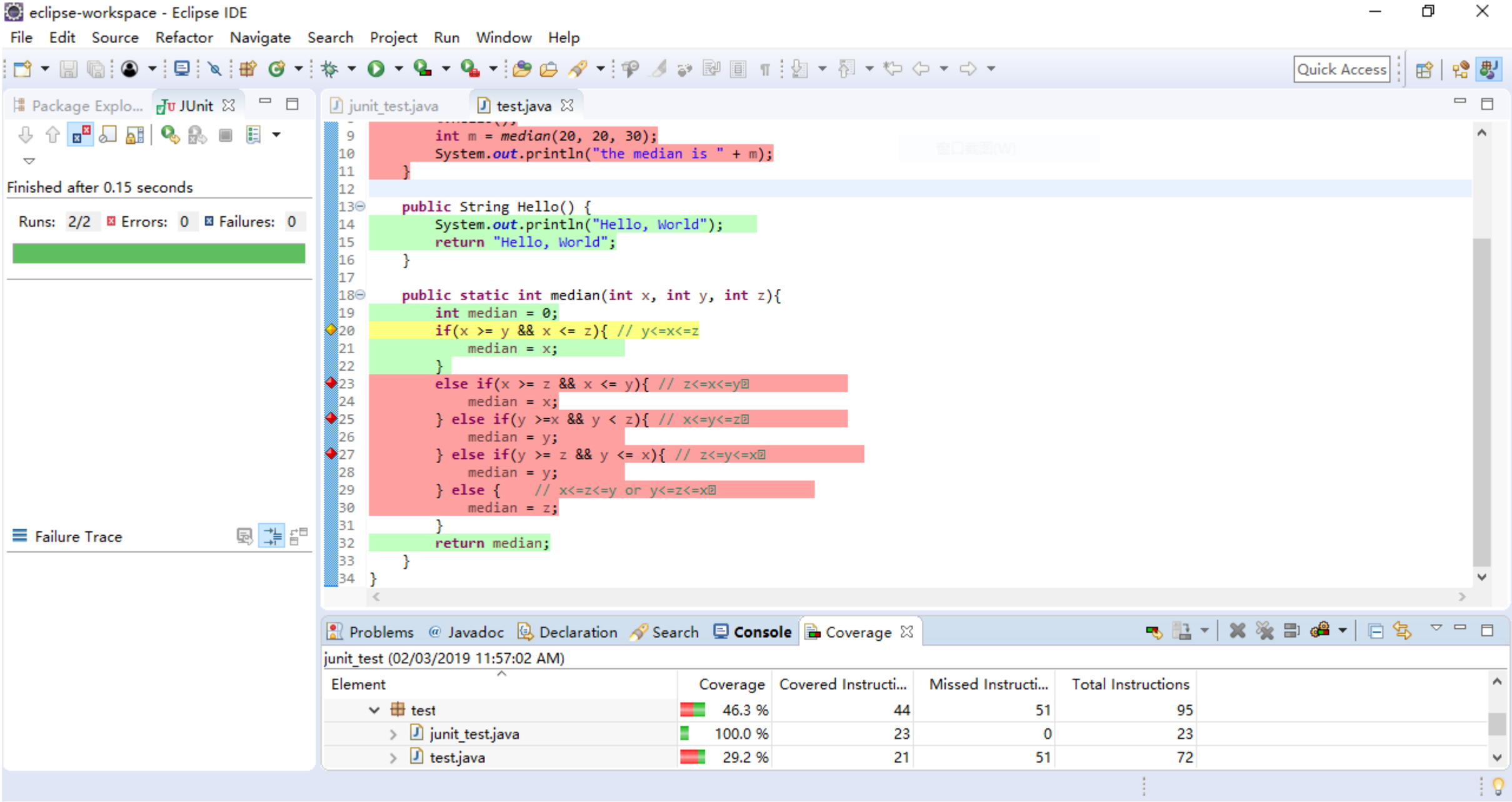
EclEmma: execution

- The installation was successful if you can see the coverage launcher in the toolbar of the Java perspective:
- Coverage collection





EclEmma: Demo



References:

- Introduction to Software Testing (1st Edition),

  ISBN: 978-0521880381

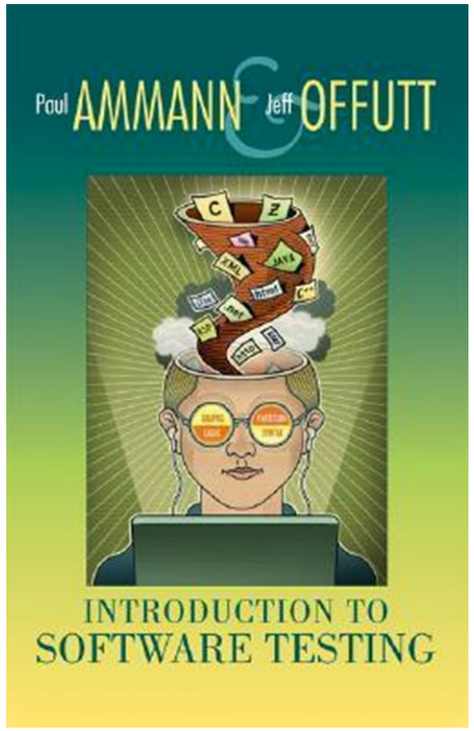
- Foundations of Software Testing (2nd Edition),

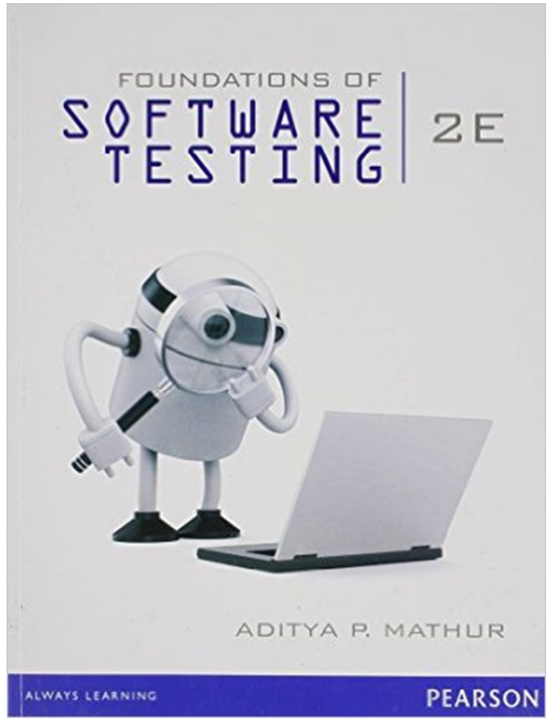
  ISBN: 978-8131794760

- K Naik and P Tripathy, Software

  Testing and Quality Assurance: Theory and

  Practice, Wiley, ISBN: 978-0-471-78911-6, 2008.





Acknowledgment:

. Dr Lingming Zhang, UT Dallas

. Dr Diana Kuo, Swinburne University of Technology

. A/Prof. Dan Hao, Peking University

JUnit – Online Resources

https://www.tutorialspoint.com/junit/junit_quick_guide.htm

http://www.vogella.com/tutorials/JUnit/article.html

https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm

www.junit.org

Prepare for testing—IntelliJ IDEA (jetbrains.com)

Thanks!


Testing
Software
design
articles
management
web
use
test
code
quality
development
good
risk
usability
Agile
testers
information
Project
People
article