UNIT TESTING

CSI3J4 #13

Tim Dosen KK SE

Sub Bahasan

- Test Case Design
- White Box Testing
- Control Structure Testing
- Basis Path Testing
- Tool Testing Otomatis

Sub Bahasan 1

Test Case Design

What is a “Good” Test?

- A good test has a high probability of finding an error
- A good test is not redundant.
- A good test should be “best of breed”
- A good test should be neither too simple nor too complex

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Internal and External Views

- Any engineered product (and most other things) can be tested in one of two ways:
  - Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;
  - **Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised.**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

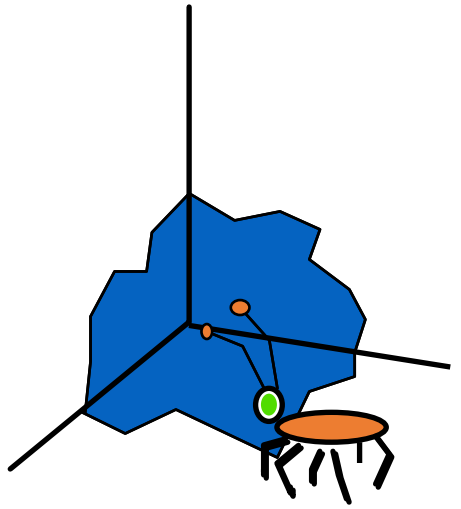
Test Case Design

OBJECTIVE **to uncover errors**

CRITERIA **in a complete manner**

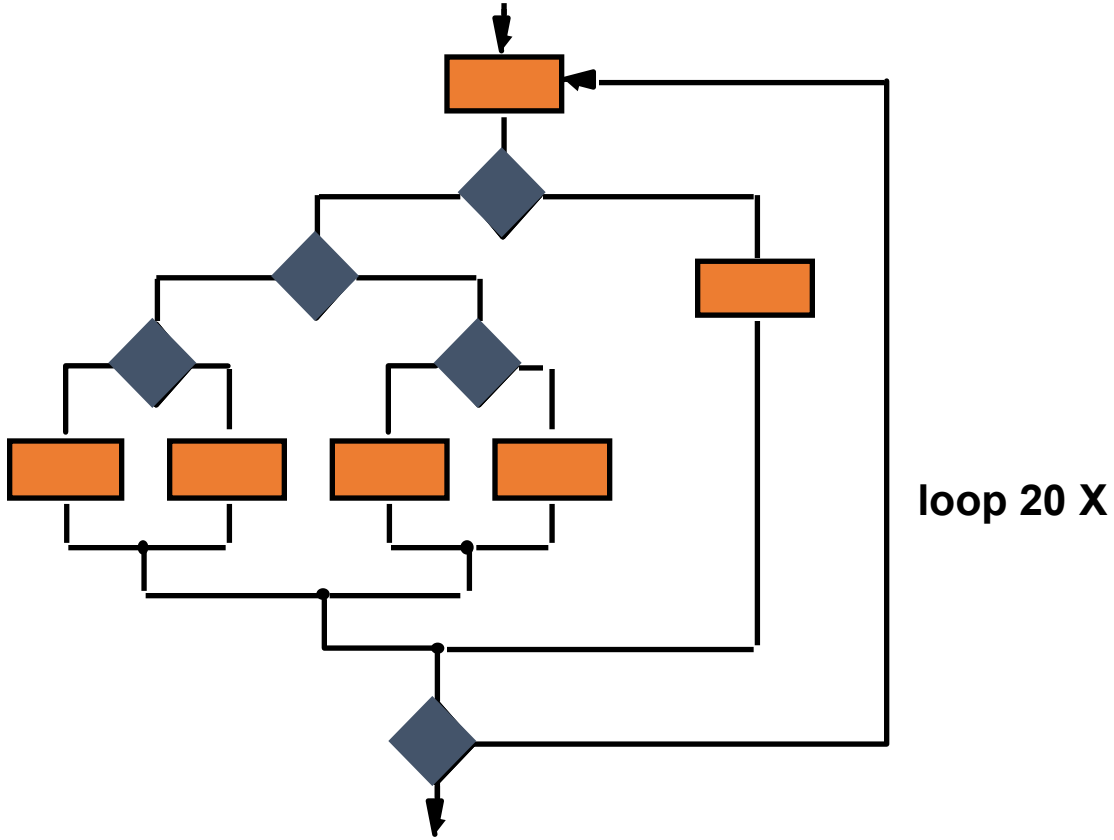
CONSTRAINT **with a minimum of effort and time**

"Bugs lurk in corners and congregate at boundaries ...“
(*Boris Beizer*)



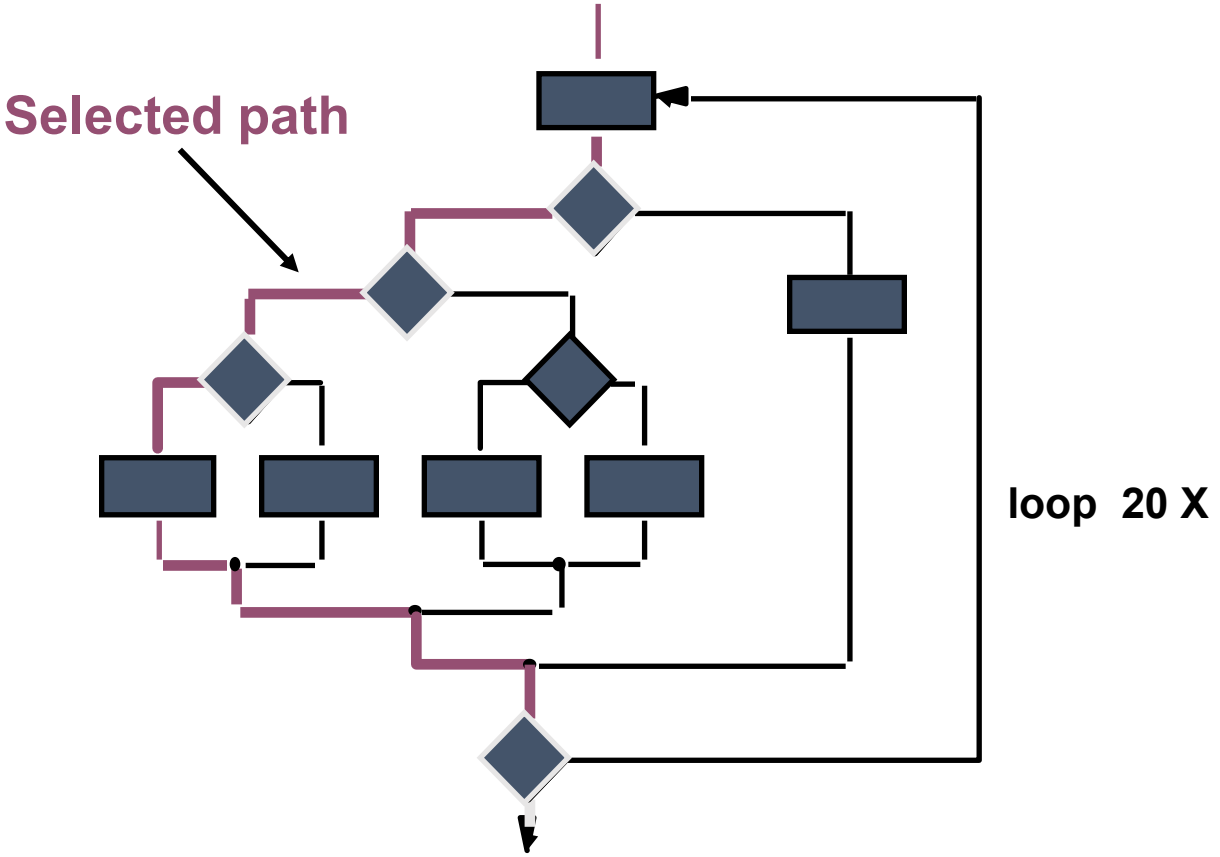
These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Exhaustive Testing



5 path need to be cheked ➔ there are 5^{20} or 95.367.400.000.000 possible paths!
If we check these path one by one, and execute one test per millisecond,
it would take 3.170 years to test this program!!

Selective Testing



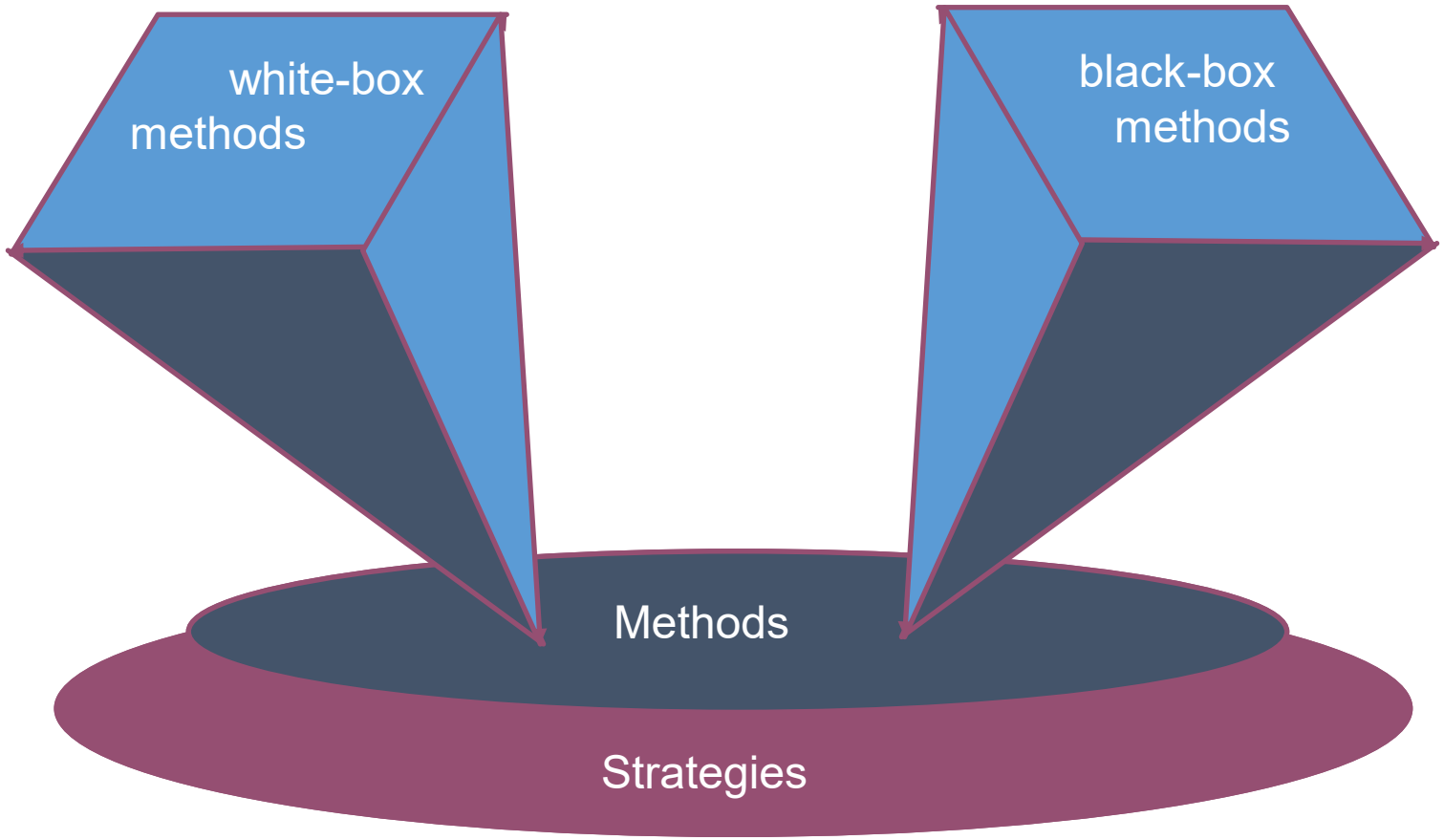
How to select test case? **How to design test case?**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

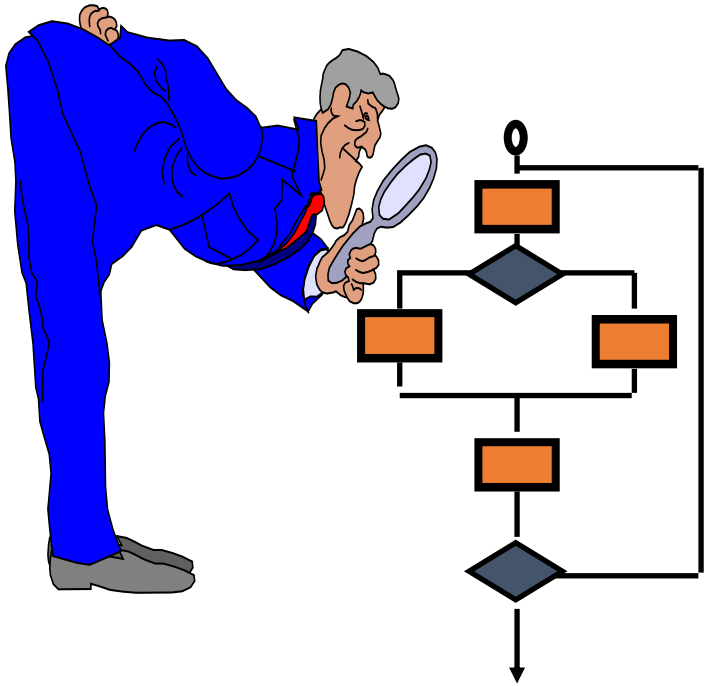
Sub Bahasan 2

White Box Testing

Software Testing



White-Box Testing



... our goal is to ensure that all statements and conditions have been executed at least once ...

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Why Cover?

- **logic errors and incorrect assumptions are inversely proportional to a path's execution probability**
- **we often believe that a path is not likely to be executed; in fact, reality is often counter intuitive**
- **typographical errors are random; it's likely that untested paths will contain some errors**

Sub Bahasan 3

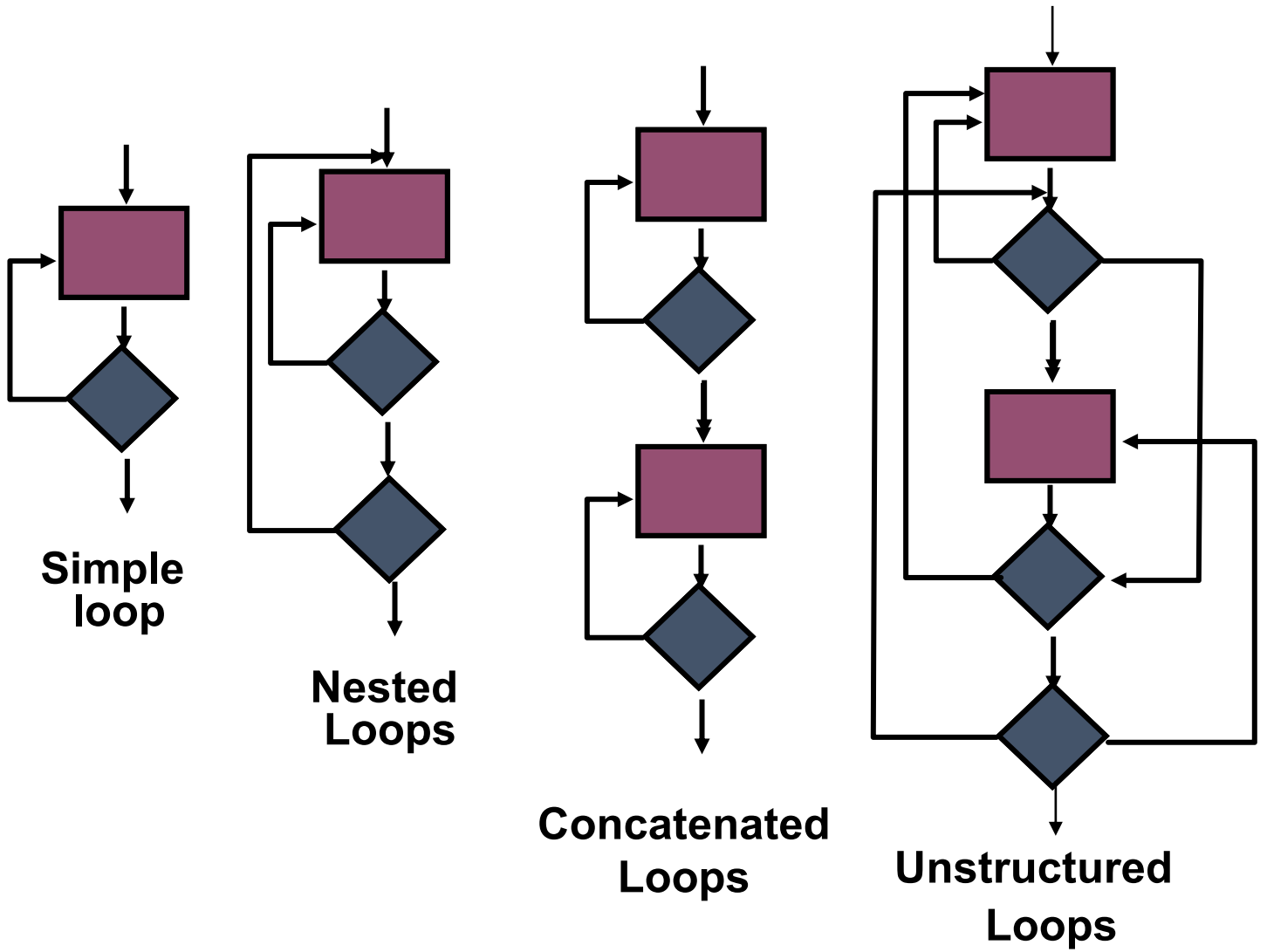
Control Structure Testing

Control Structure Testing

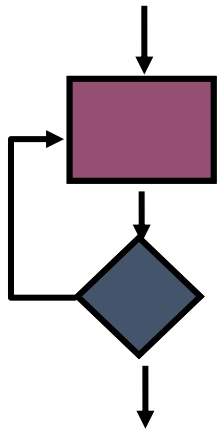
- Condition testing — a test case design method that exercises the logical conditions contained in a program module
- Data flow testing — selects test paths of a program according to the locations of definitions and uses of variables in the program

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Loop Testing, how to test these loops?



Simple Loops



Simple loop

Possible conditions—Simple Loops

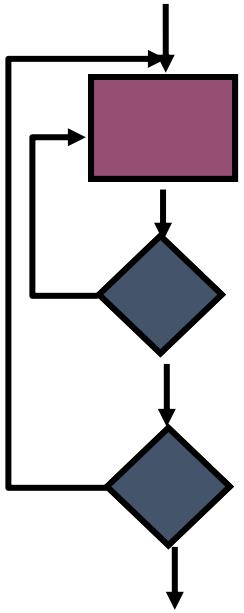
1. **skip the loop entirely**
2. **only one pass through the loop**
3. two passes through the loop
4. m passes through the loop m < n
5. (n-1), n, and (n+1) passes through the loop

where n is the maximum number of allowable passes

It's enough to check no 1 and no 2

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Nested



Nested Loops

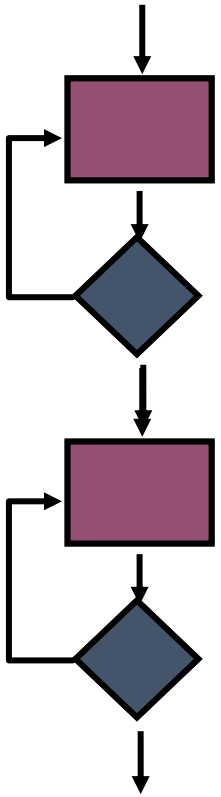
Nested Loops

Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.

Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.

Move out one loop and set it up as in step 2, holding all other loops at typical values. Continue this step until the outermost loop has been tested.

Concatenated Loop



Concatenated Loops

Concatenated Loops

If the loops are independent of one another
then treat each as a simple loop
else* treat as nested loops
endif*

for example, the final loop counter value of loop 1 is used to initialize loop 2.

Data Flow Testing

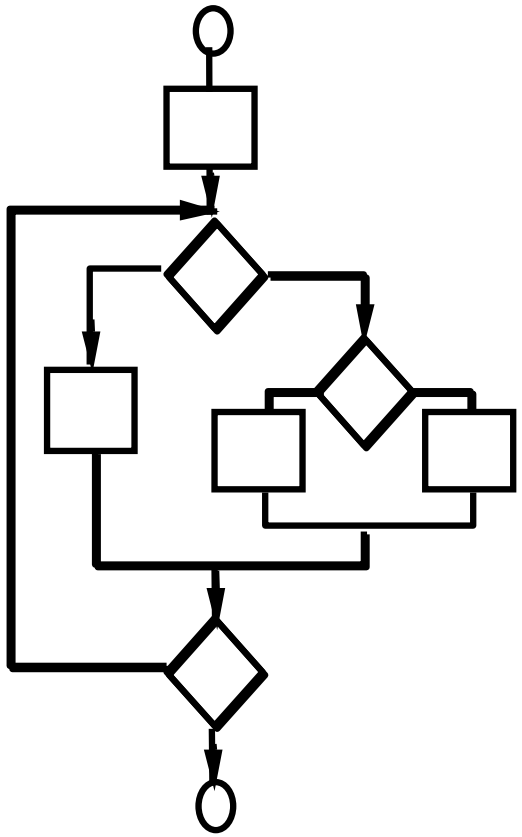
- The data flow testing method [Fra93] selects test paths of a program according to the locations of definitions and uses of variables in the program.
  - Assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables. For a statement with *S* as its statement number
    - DEF(*S*) = {*X* | statement *S* contains a definition of *X*}
    - USE(*S*) = {*X* | statement *S* contains a use of *X*}
  - A *definition-use (DU) chain* of variable X is of the form [*X*, *S*, *S'*], where *S* and *S'* are statement numbers, *X* is in DEF(*S*) and USE(*S'*), and the definition of *X* in statement *S* is live at statement *S'*

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Sub Bahasan 4

Basis Path Testing

Basis Path Testing



First, we compute the cyclomatic complexity:

number of simple decisions + 1

or

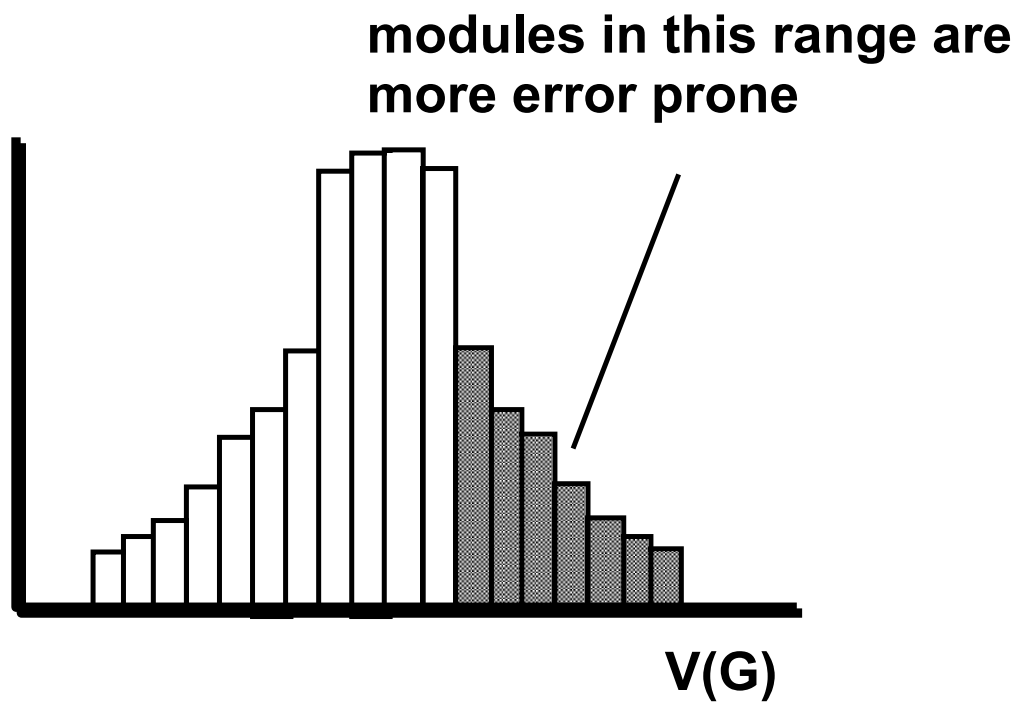
number of enclosed areas + 1

In this case, V(G) = 4

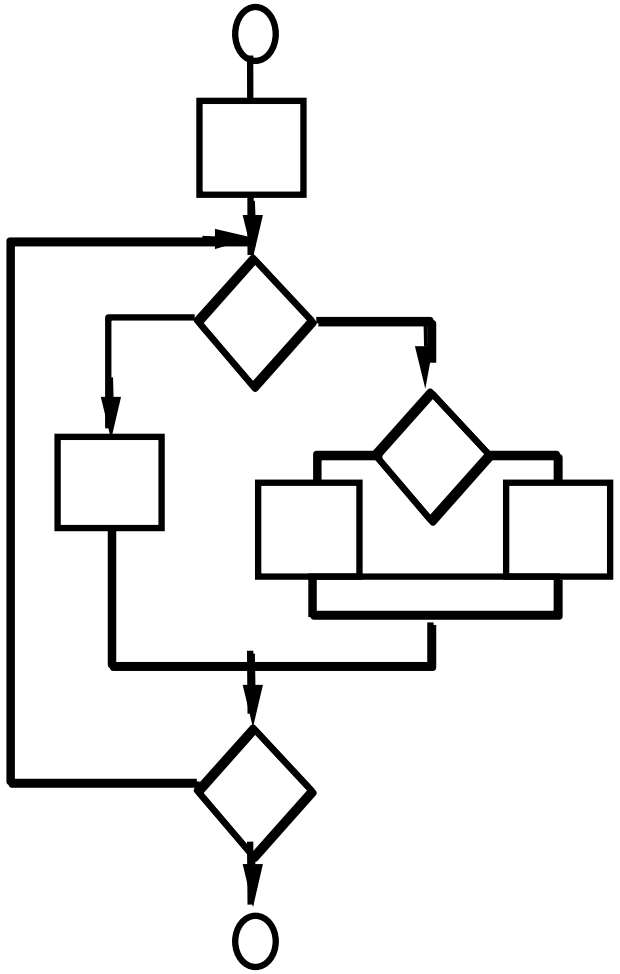
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Cyclomatic Complexity

A number of industry studies have indicated that the higher V(G), the higher the probability or errors.



Basis Path Testing



Next, we derive the independent paths:

Since V(G) = 4,
there are four paths

Path 1: 1,2,3,6,7,8

Path 2: 1,2,3,5,7,8

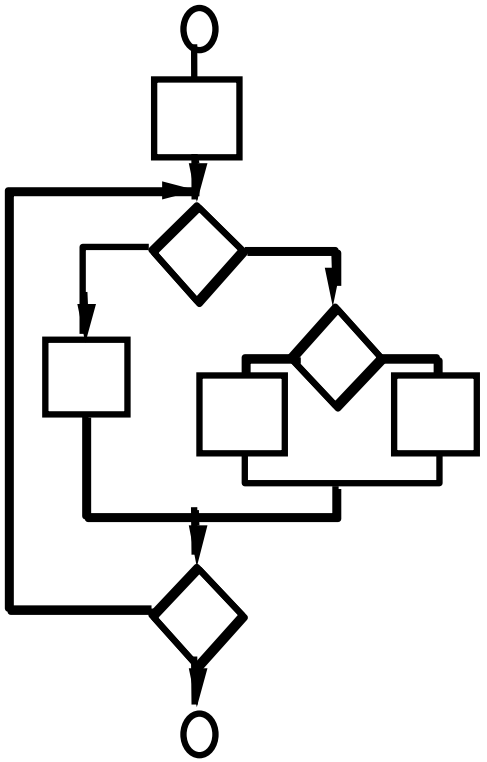
Path 3: 1,2,4,7,8

Path 4: 1,2,4,7,2,4,7,8

Finally, we derive test cases to exercise these paths.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Basis Path Testing Notes



- **you don't need a flow chart, but the picture will help when you trace program paths**
- **count each simple logical test, compound tests count as 2 or more**
- **basis path testing should be applied to critical modules**

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Deriving Test Cases

- Using the design or code as a foundation, draw a corresponding flow graph (untuk mudahnya flow graph = flow chart dengan symbol lingkaran untuk setiap activitynya).
- Determine the cyclomatic complexity of the resultant flow graph.
- Determine a basis set of linearly independent paths.
- Prepare test cases that will force execution of each path in the basis set.

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

Contoh sederhana

```
Read (usn);
Read (pwd);
if (usn = “admin”) and (pwd = “123”){
        MenuUtama();
}
Else {
        write (“Username atau password salah”);
}
```

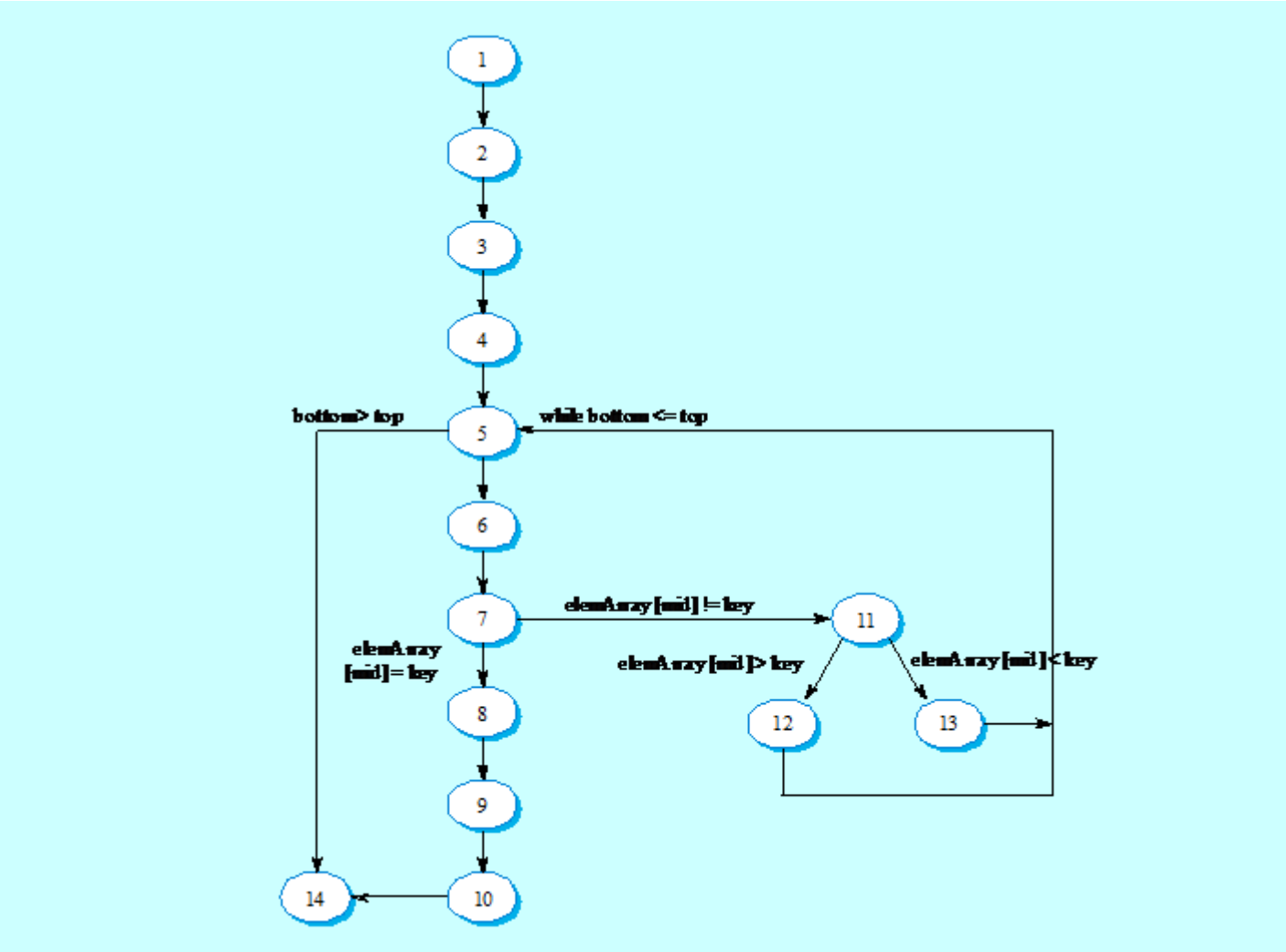
PENDAPAT #1
Cyclomatic complexity ?
: jumlah kondisi + 1
: 2 + 1
= 3

Siapkan 3 kasus uji :
1. **usn = “admin” & pwd = “123”**
2. **usn = “admin” & pwd selain “123”**
3. **usn selain admin & pwd = “123”**

PENDAPAT #2
Cyclomatic complexity ?
: jumlah kondisi + 1
: 1 (karena pakai AND) + 1
= 2

Siapkan 2 kasus uji :
1. **usn = “admin” & pwd = “123”**
2. **selainnya**

Case Study : Binary search flow graph



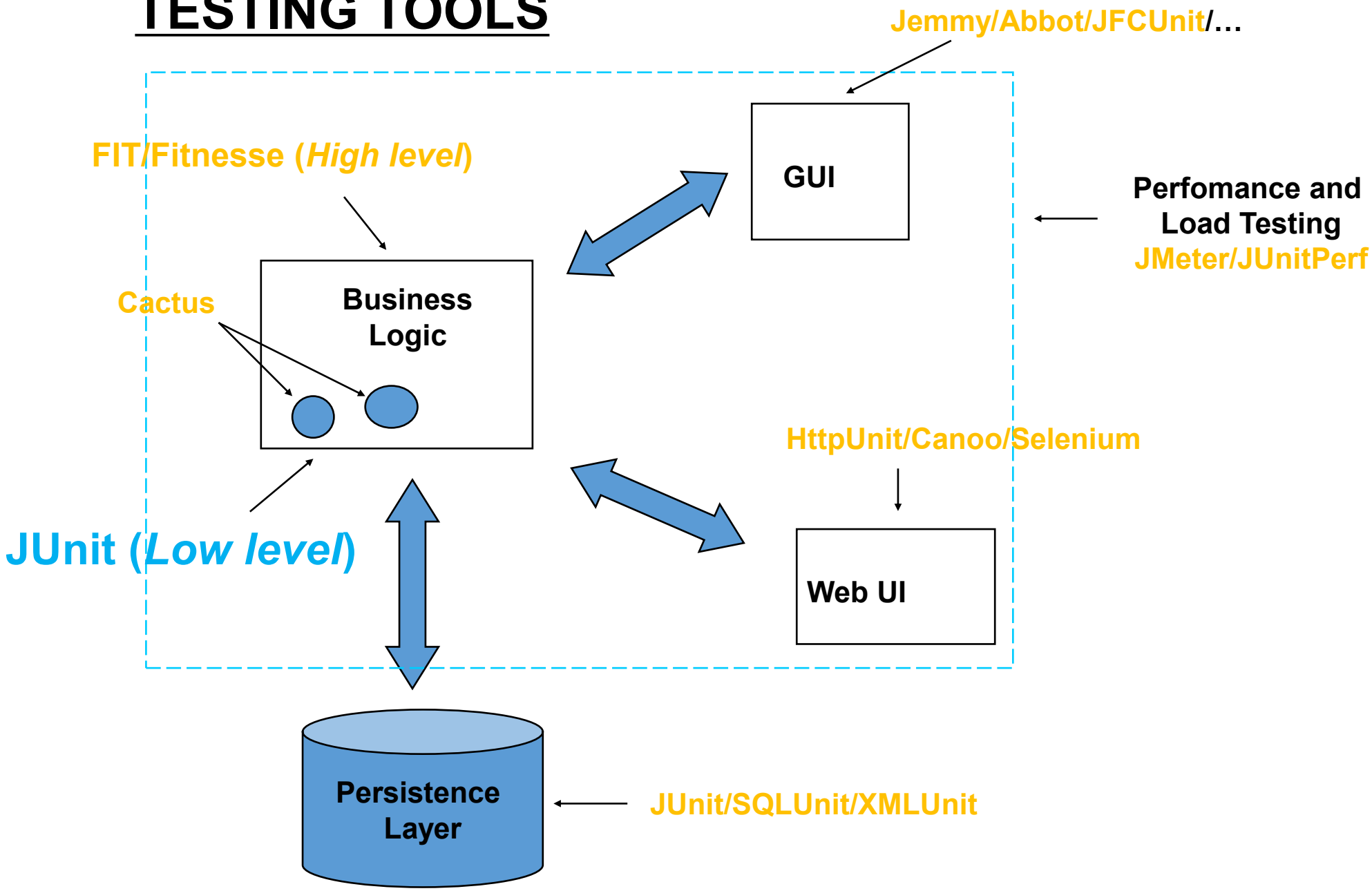
Independent paths

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
- Test cases should be derived so that all of these paths are executed
- A dynamic program analyser may be used to check that paths have been executed

Sub Bahasan 5

Tool Testing Otomatis

TESTING TOOLS



What should be tested?

» Develop automated tests for

- GUI objects and events
- Application functions
- Application special features
- Application performance and scalability
- Application reliability
- Application compatibility
- Application performance
- Database verification

JUnit

- JUnit is a framework for writing tests
  - JUnit was written by Erich Gamma (of Design Patterns fame) and Kent Beck (creator of XP methodology)
  - JUnit uses Java's reflection capabilities (Java programs can examine their own code)
  - JUnit helps the programmer:
    - define and execute tests and test suites
    - formalize requirements and clarify architecture
    - write and debug code
    - integrate code and always be ready to release a working version
  - JUnit is not yet (as far as I know) included in Sun's SDK, but an increasing number of IDEs include it
  - BlueJ, JBuilder, and Eclipse now provide JUnit tools

Terminology

- A test fixture sets up the data (both objects and primitives) that are needed to run tests
  - Example: If you are testing code that updates an employee record, you need an employee record to test it on
- A unit test is a test of a *single* class
- A test case tests the response of a single method to a particular set of inputs
- A test suite is a collection of test cases
- A test runner is software that runs tests and reports results

- An integration test is a test of how well classes work together
  - JUnit provides some limited support for integration tests

Example : Class Counter

```
public class Counter {
        int count = 0;

        public int increment() {
            return ++count;
        }

        public int decrement() {
            return --count;
        }

       public int getCount() {
           return count;
       }
}
```

- Bagaimana menguji class ini?
- Biasanya buat objek di program utama

```
Public static void main (String args[]){
            Counter c = new Counter();
            System.out.println(c.count);
            System.out.println(c.increment());
            System.out.println(c.decrement());
            System.out.println(c.getCount());
}
```

- Bagaimana menguji dengan Junit?

Structure of a JUnit test class

- We want to make class for test Counter
- public class CounterTest
  extends junit.framework.TestCase {
  - This is the unit test for the Counter class; it declares (and possibly defines) values used by one or more tests
- public CounterTest() { }
  - This is the default constructor
- protected void setUp()
  - Creates a test fixture by creating and initializing objects and values
- protected void tearDown()
  - Releases any system resources used by the test fixture
- public void testIncrement(), public void testDecrement().

JUnit tests for **Counter**

```
public class CounterTest extends junit.framework.TestCase {
    Counter counter1;

    public CounterTest() { }    // default constructor

    protected void setUp() {    // creates a (simple) test fixture
        counter1 = new Counter();
    }

    protected void tearDown() { } // no resources to release

    public void testIncrement() {// periksa apakah fungsi increment jalan & benar

        assertTrue(counter1.increment() == 1);

        assertTrue(counter1.increment() == 2);
     }

    public void testDecrement() {// periksa apakah fungsi decrement jalan & benar
        assertTrue(counter1.decrement() == -1);
    }
}
```

Creating a Test Class for java program in Netbean

1. Right-click java program (ex : Counter.java) and choose Tools > Create Tests.
2. Modify the name of the test class to CounterUnitTest in the Create Tests dialog.
3. Select JUnit in the Framework dropdown list.
4. Deselect Test Initializer and Test Finalizer. Click OK.

TUGAS

Dari 3 bilangan a,b,c segitiga apa dapat dibangun?

Seorang profesor software testing mengatakan “Belum seru bicara software testing, jika belum membahas segitiga.

- Dari masukan 3 bilangan a, b, c bebas segitiga apa dapat dibangun.
- Jika ada yang negatif atau 0, tidak ada segitiga dapat dibangun.
- Jika bilangan yang terbesar lebih besar atau sama dengan penjumlahan dua bilangan lainnya yang lebih kecil, tidak ada segitiga dapat dibangun.
- Jika a=b atau b=c atau a=c (namun tidak sama dengan salah satu yang lain maka segitiga SAMA KAKI.
- Jika a=b dan b=c maka segitiga SAMA SISI.
- Jika kuadrat bilangan terbesar = penjumlahan dari kuadrat dua bilangan lainnya, maka SEGITIGA SIKU-SIKU.
- Jika bukan sama kaki, sama sisi atau siku-siku, namun bilangan terbesarnya lebih kecil daripada penjumlahan dua bilangan lainnya, maka SEGITIGA BEBAS.
- Kasus 1 angka-angkanya bulat,
- Kasus 2 angka-angkanya pecahan (misal : masukan 2.99, 3.01, 3.00 akan dianggap segitiga sama sisi
- Buatkan codenya dan buatkan rancangan pengujiannya mengikut langkah2 di slide 25