

# Structured testing

Testing is a process to execute the program to find out mistakes.

# Test

- Own collection of input data and conditions to execute the program;
- A set of expected program's results.

# Testing

Exhaustive testing –

complete check of the program

A good test case –

a test case with a high probability of error detection.

# Testing

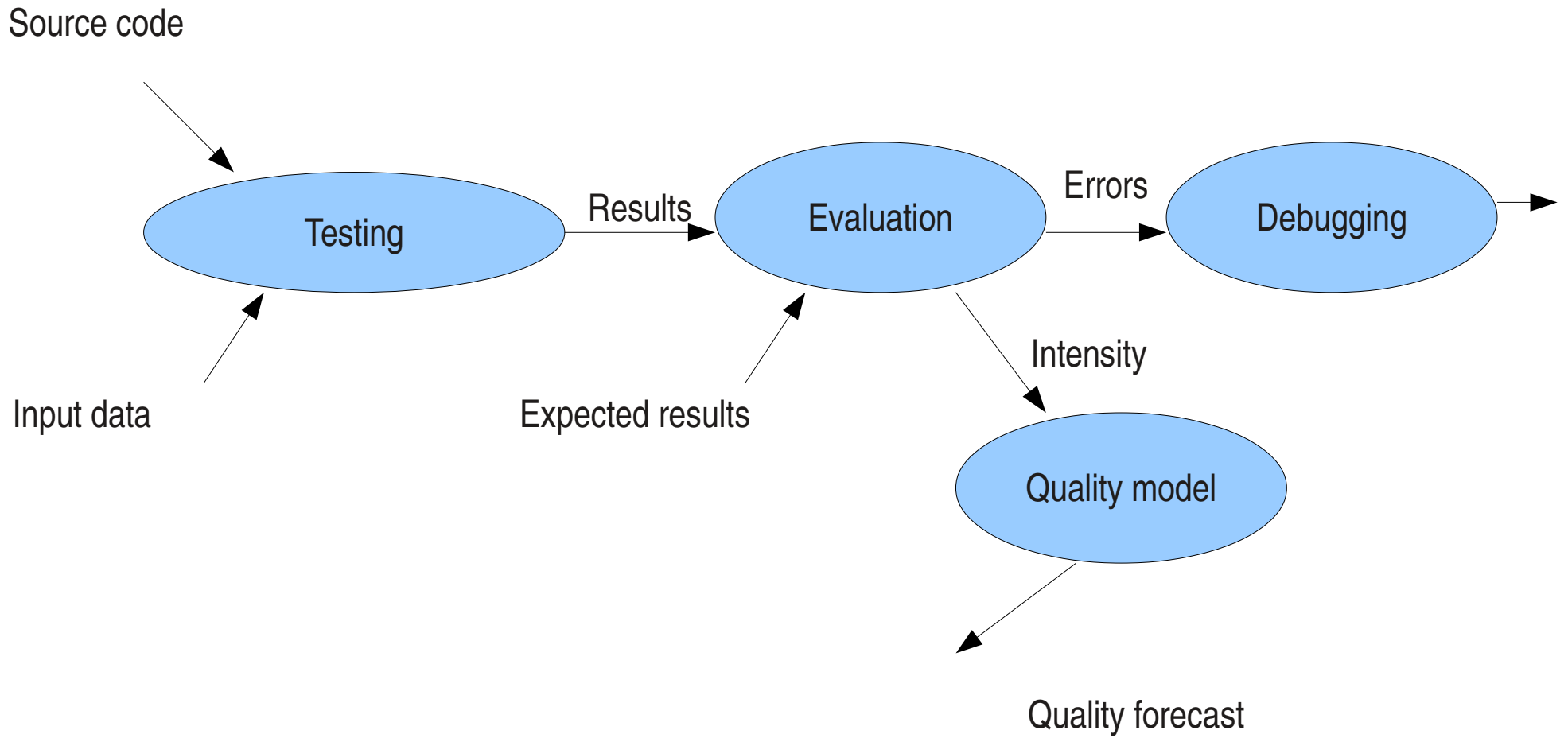
## **CAN**

- Find errors;
- Demonstrate of compliance functions;
- Demonstrate of requirements satisfaction;
- Show quality.

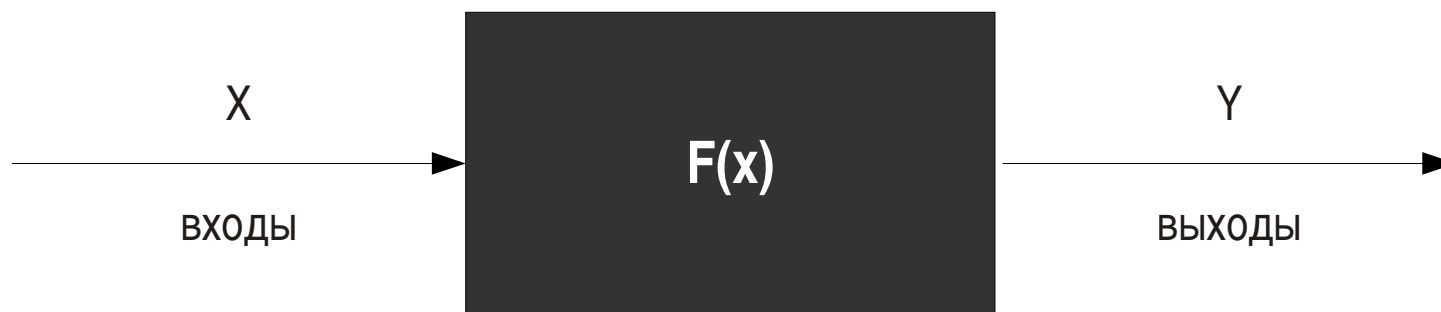
## **CAN NOT**

Show absence of errors.

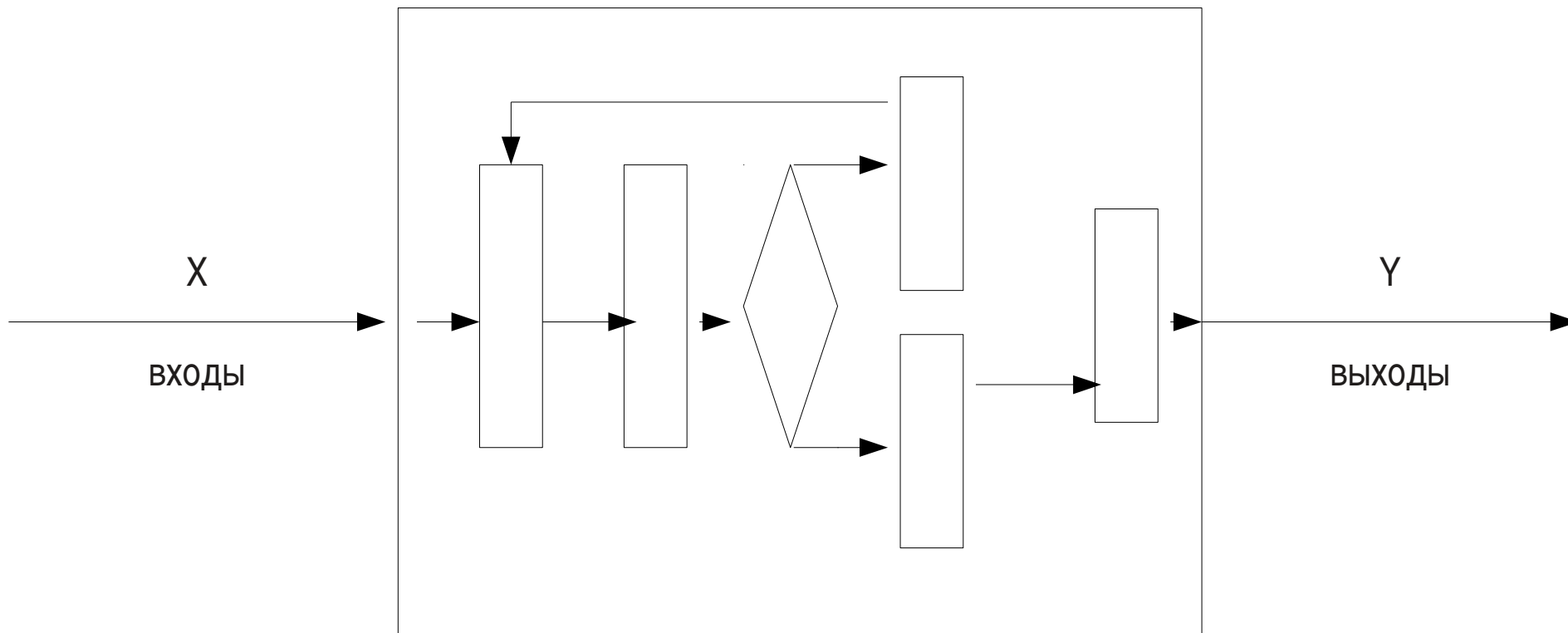
# Testing



# «Black box»



# «White box»



# Testing of base path

Том МакКейб, 1976

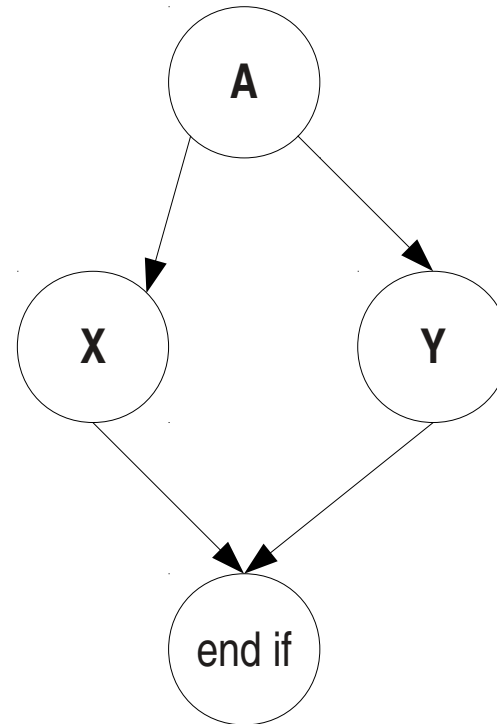
## Flow graph

- 1) Demonstrate control structure;
- 2) Node = linear flow;
- 3) Arc = control flow;
- 4) Operator and predicate nodes;
- 5) Predicate node = simple condition;
- 6) Regions;
- 7) Environment.



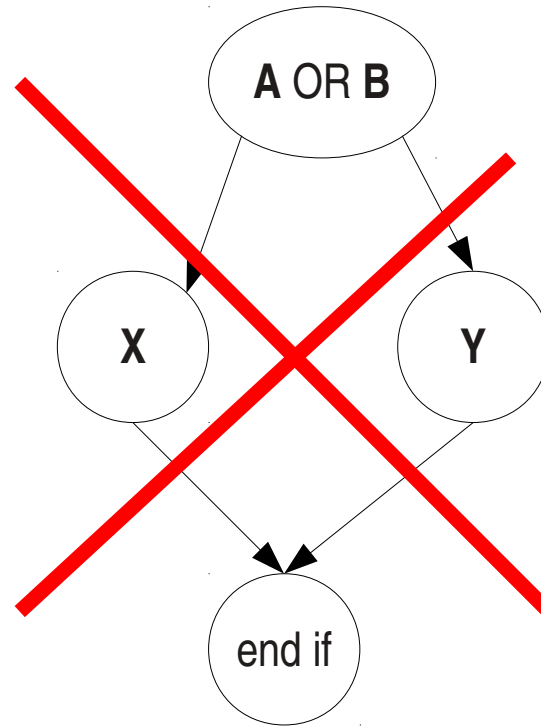
# Testing of base path

If a  
then x  
else y  
end if



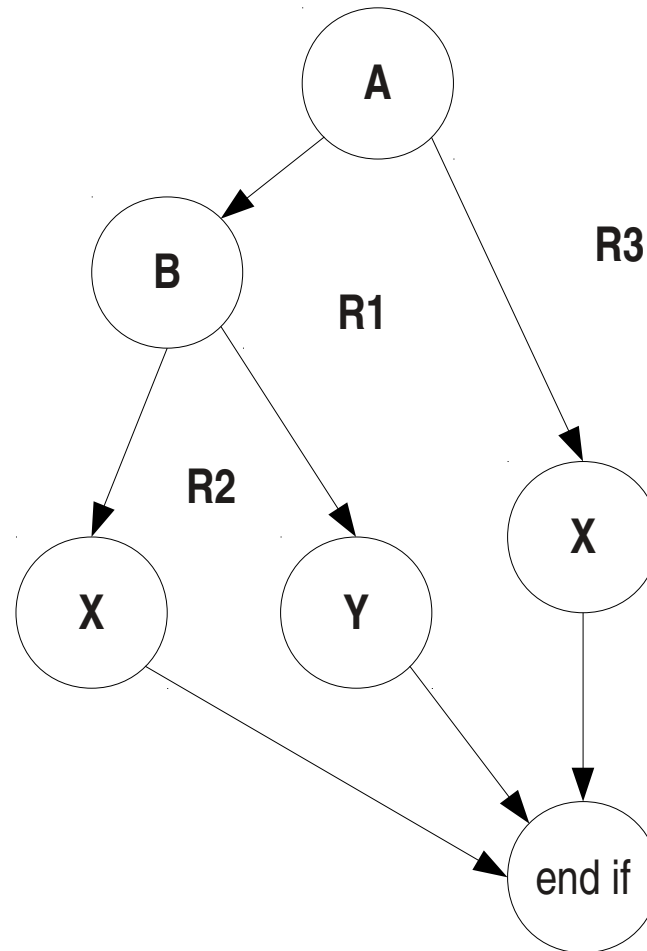
# Testing of base path

If a OR b  
then x  
else y  
end if



# Testing of base path

If a OR b  
then x  
else y  
end if



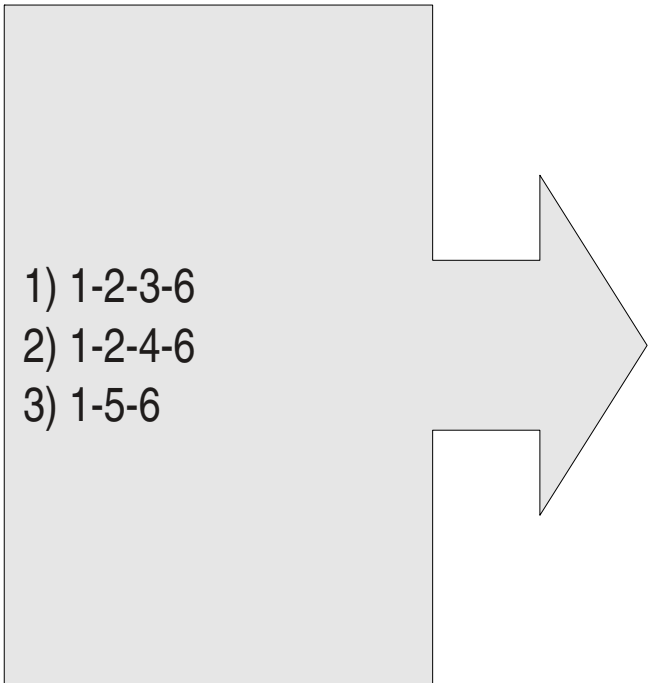
# Cyclomatic complexity

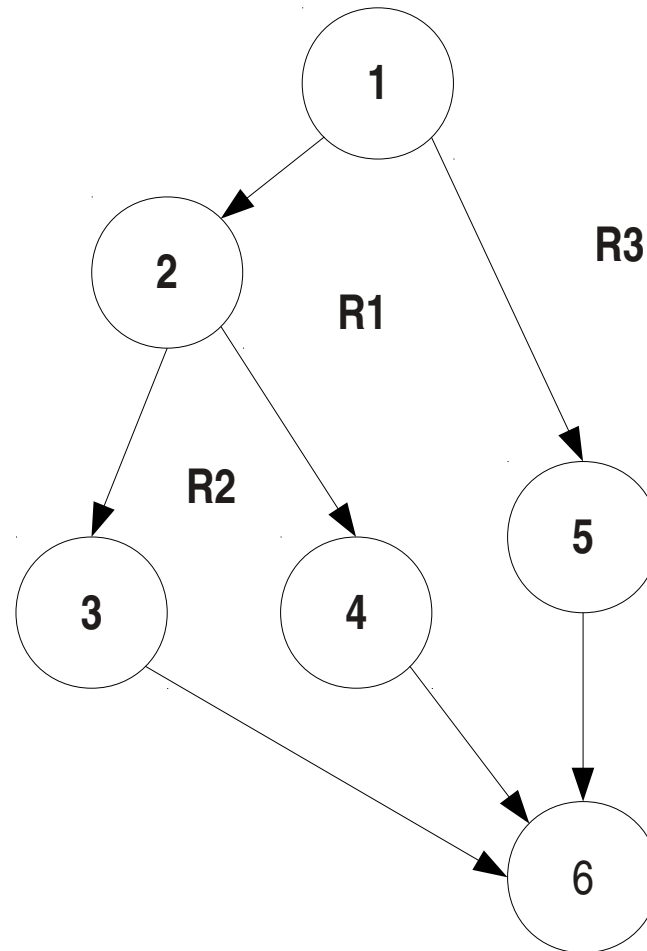
Cyclomatic complexity defines:

- The number of independent paths in the base set of the algorithm;
- The upper estimate of the number of tests that guarantees executed once all operators.

All independent paths form the base set.

# Cyclomatic complexity

- 
- 1) 1-2-3-6
  - 2) 1-2-4-6
  - 3) 1-5-6



# Cyclomatic complexity

How to calculate:

1. Number of regions;
2.  $V(G)=E-N+2$ ,  $E$  — number of arcs,  $N$  — number of nodes;
3.  $V(G)=p+1$ , где  $p$  — number of predicate nodes.

# Testing conditions

The goal - the creation of test cases for testing logical conditions of the program.

Simple condition - a boolean expression or expression of relations.

Expression of relations:  $E1 \langle \text{operator} \rangle E2$ ,  
E1, E2 — arithmetic expression,  
operator -  $\langle, \rangle, =, \langle \rangle, \langle =, \rangle =$

# Testing conditions

Complex condition consists of a few simple conditions, boolean operators, and parentheses.

Boolean operators — AND, OR, NOT.

Conditions that do not contain expressions of relations are called boolean expression



# Testing conditions

## Elements of condition:

- Boolean operator;
- Boolean variable;
- Pair of braces;
- Expression of relations;
- Arithmetic expression.

# Testing conditions

Type of erros:

- Error of boolean operator (incorrect, missed, excessed);
- Error of boolean variable;
- Error of pair of braces;
- Error of expression of relations;
- Error of arithmetic expression.

# Testing conditions

## Advantages

- It is easy to take a measurement of test coverage conditions;
- Test coverage - the foundation for the generation of additional tests.

## Types of methods of testing conditions:

- Testing of the branches;
- Testing dataflow.

# Testing of the branches

This method allows to detect errors and branching operators relations at a time when restrictions are performed:

- All boolean variables, and relational operators are included in the condition only once;
- There is no shared variables in the condition.

# Testing of the branches

Complex condition C of n simple conditions makes Condition Restriction (CR)

$$CR_c = (d_1, d_2, d_3, \dots, d_n)$$

$$d_i = (true, false)$$

$$d_j = (<, >, =)$$

# Testing of the branches

For Condition Restriction (CR) we must Restriction Set (RS), whose elements are a combination of all possible values of  $d$

# Testing of the branches

$b \& (x > y) \& a$

$(\text{true}, >, \text{true})$

$(\text{false}, >, \text{true})$

$(\text{true}, <, \text{true})$

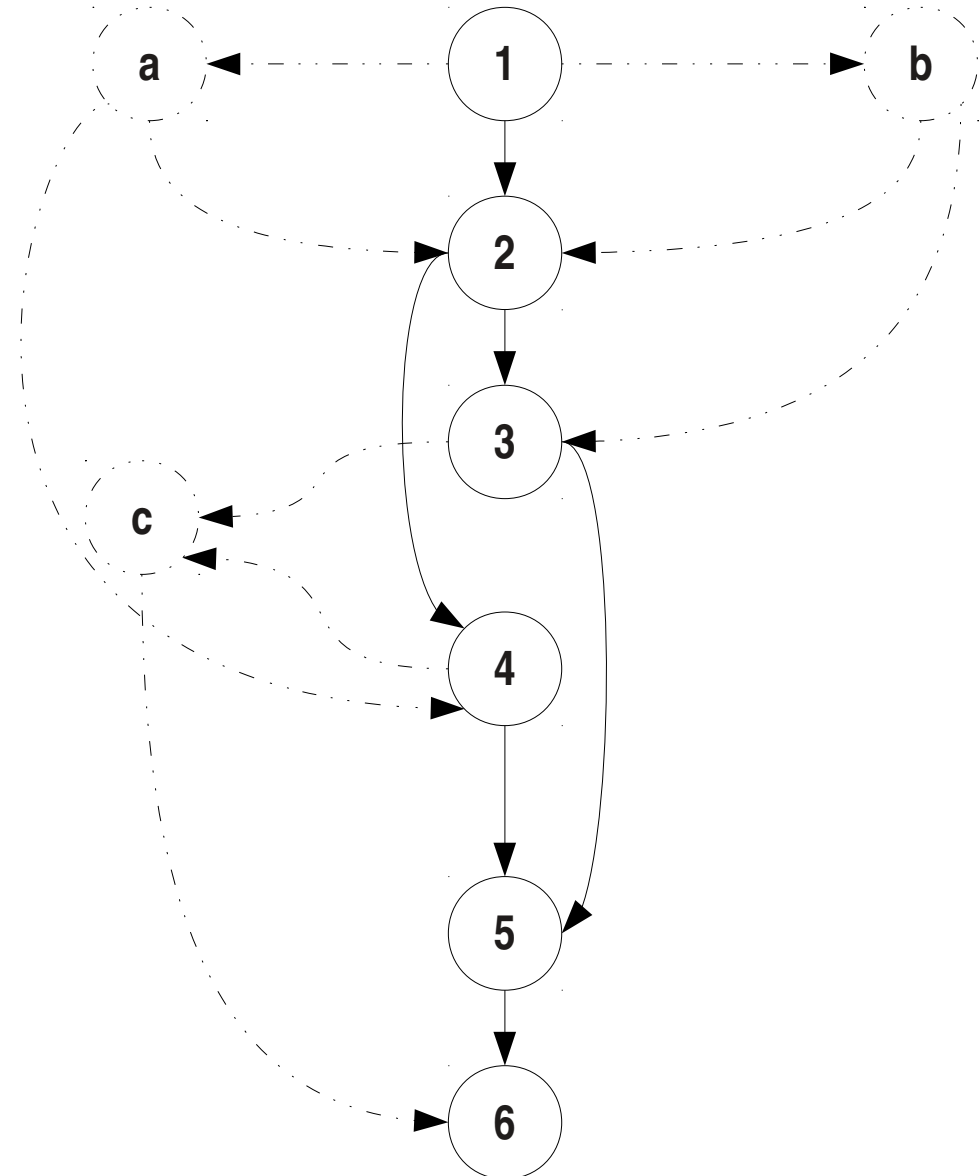
$(\text{true}, >, \text{false})$

....

$RS = \{(\text{true}, >, \text{true}), (\text{false}, >, \text{true}), \text{и т.д.}\}$

# Testing dataflow

```
a=1;  
b=2;  
if(a<b)  
{  
    c=b;  
}  
else  
{  
    c=a;  
}  
print c;
```





# Testing dataflow

Data definition - an action that changes the data element.

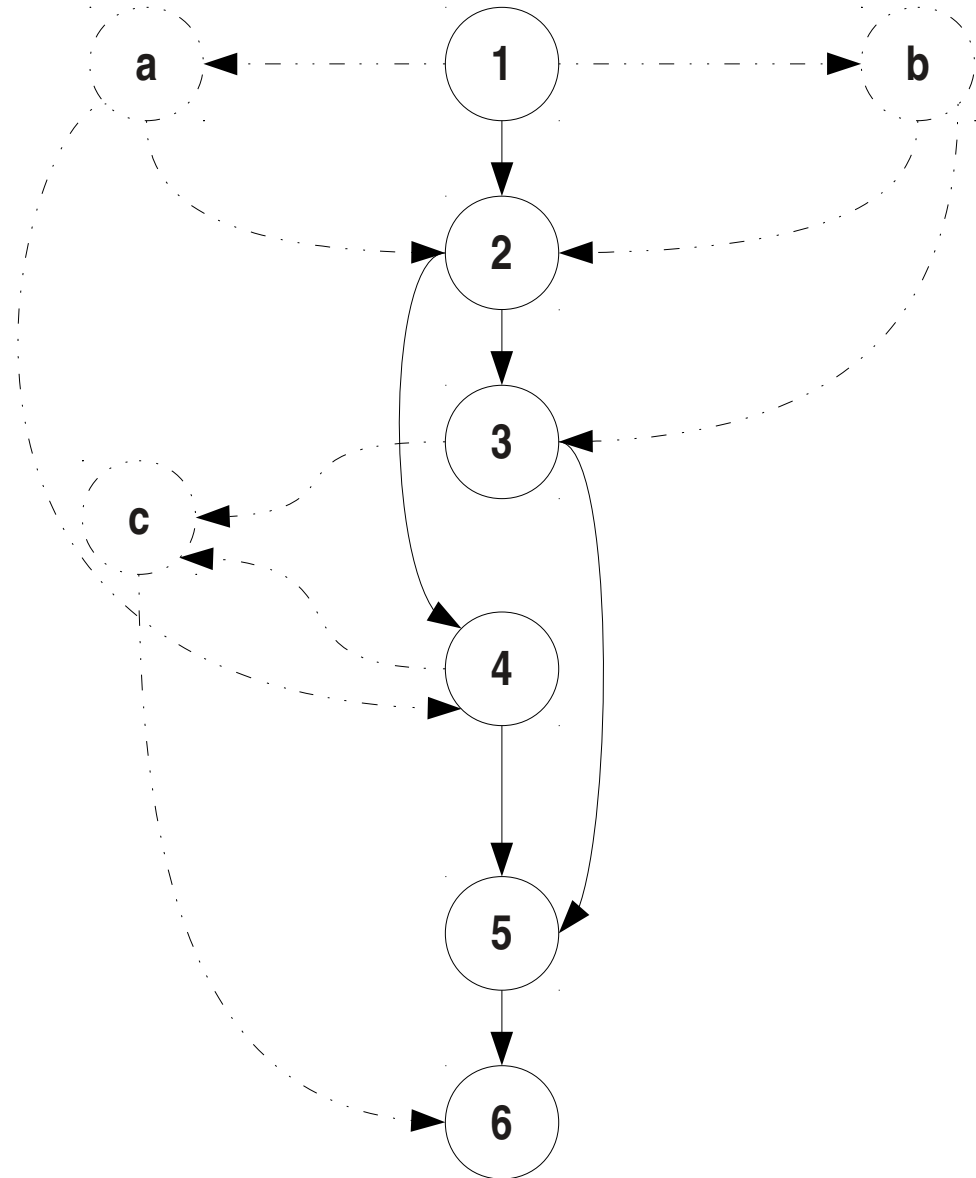
$x=f(\dots)$

Using data - the use of an element in terms of the right

side.  $\dots=f(x)$

DU-chain — chain of data definition and using.  $[x,i,j]$

# Testing dataflow



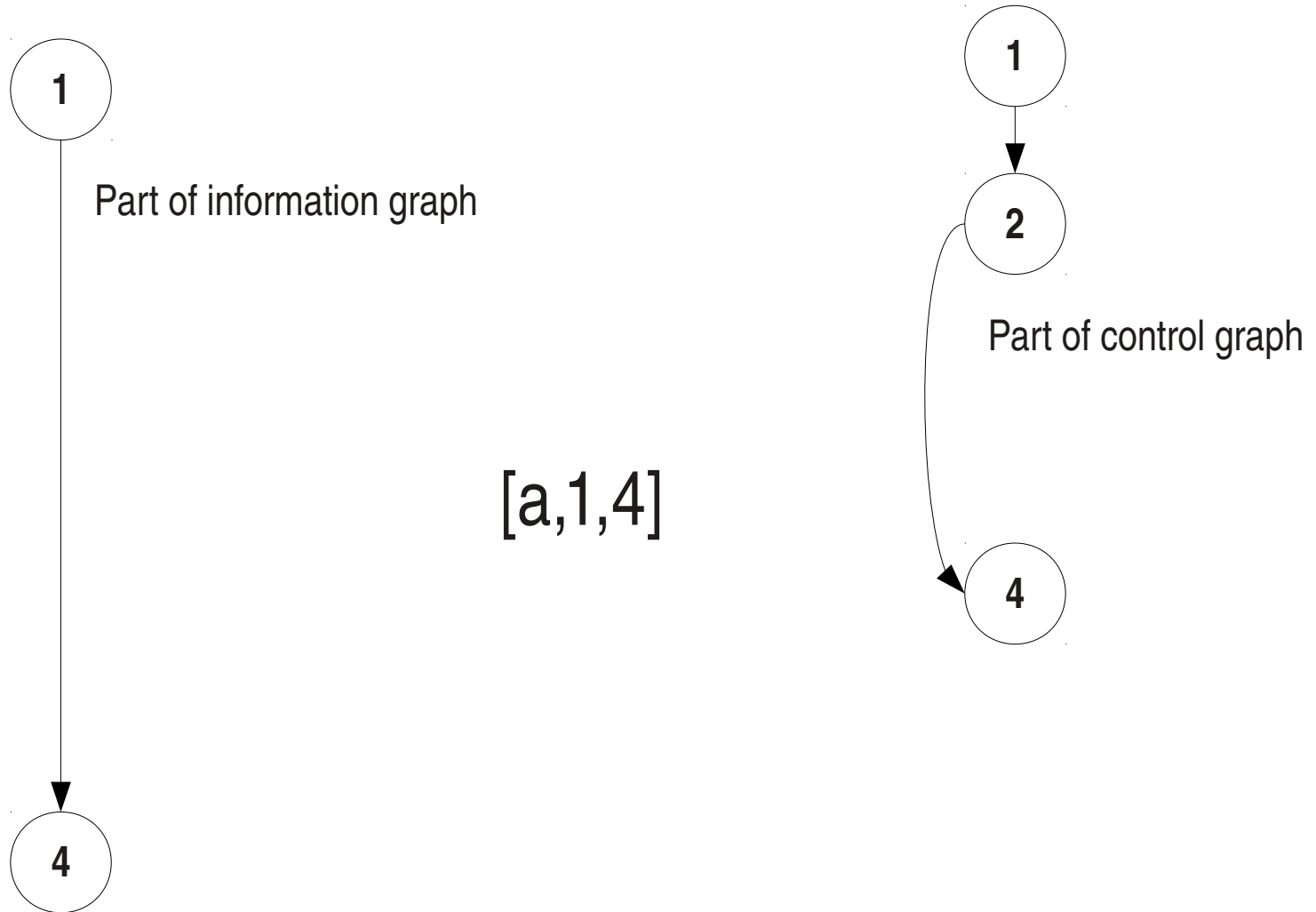
[a,1,2],[a,1,4],  
[b,1,2],...

# Testing dataflow

Steps of DU-testing:

1. Construction of the control graph;;
2. Building information graph;
3. Forming a complete set of DU-chains;
4. Forming a complete set of segments of paths in a flow graph;
5. Route building - full paths on the control column, covering a set of line segments;
6. Preparation of test cases.

# Testing dataflow



# Testing dataflow

## Advantages:

- Easy necessary analysis of the program structure;
- Ease of automation.

## Disadvantages:

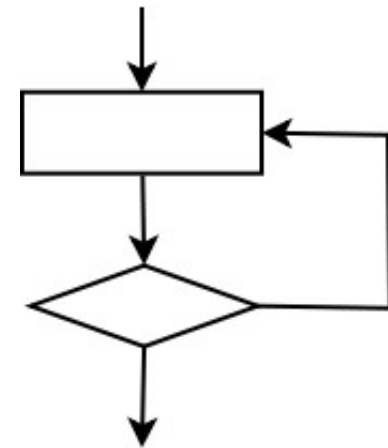
- The difficulty in selecting the most effective minimum number of tests.

# Testing cycles

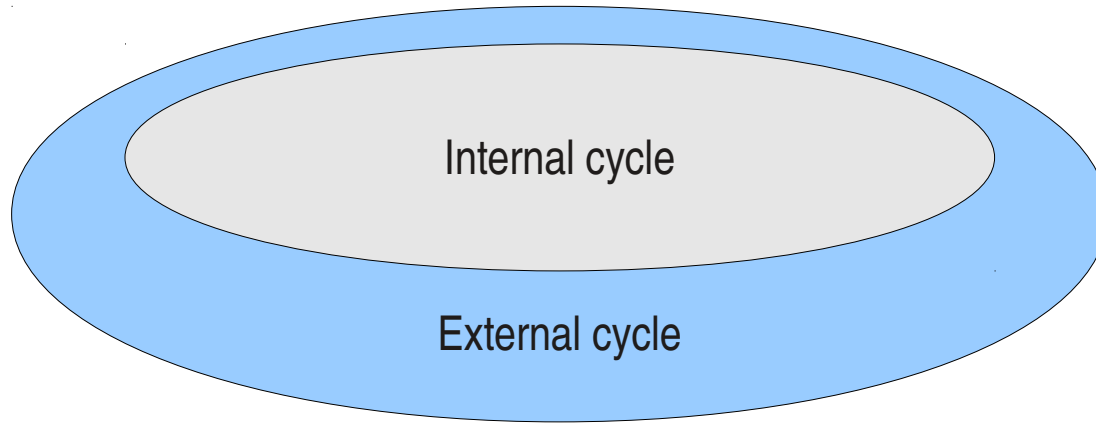
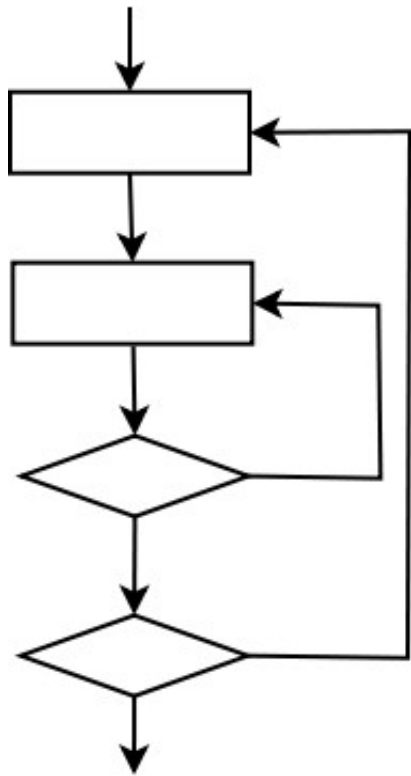
## *Simple cycle*

To test simple cycles with  $n$ -repetitions we may use one of the following sets of tests:

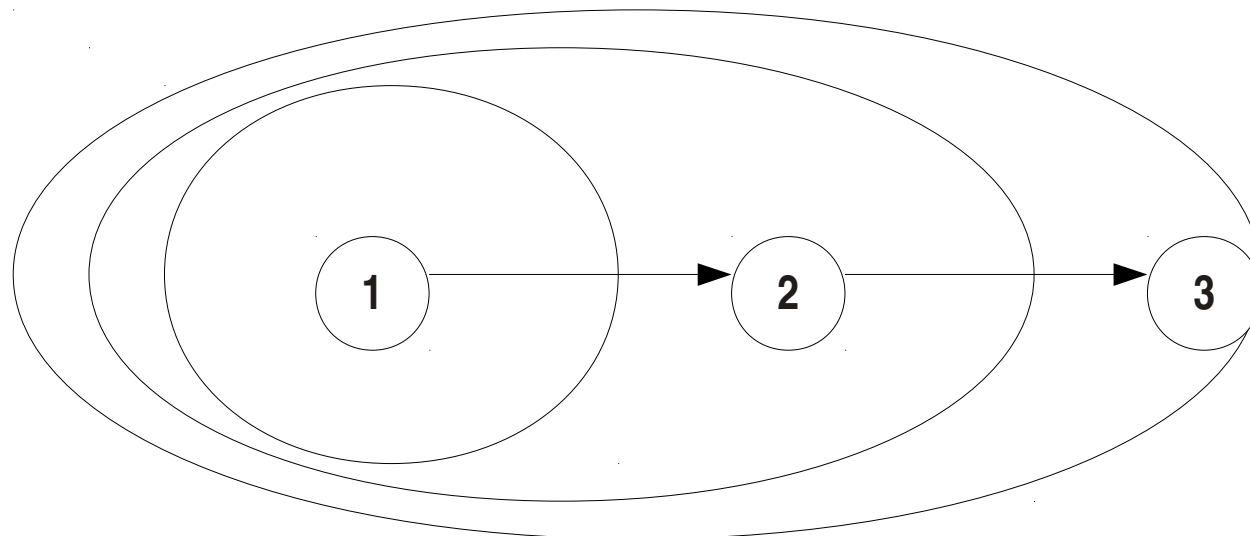
1. Run the whole cycle;
2. Only one run cycle;
3. Two runs of cycle;
4.  $m$  runs of cycle,  $m < n$ ;
5.  $n-1, n, n+1$  runs of cycle.



# Testing cycles



*Nested cycles*



# Testing cycles

## *Nested cycles*

### Steps of testing:

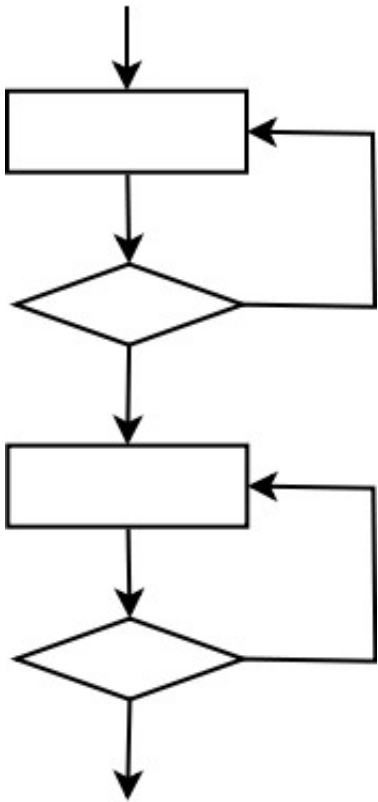
1. Select the inner cycle. Put the minimum parameters of all other cycles.
2. Making simple cycle tests for the inner cycle. Added tests for singular values and the values that go beyond the normal operating range.
3. We pass a higher level. For a nested cycle set for typical use. Test the cycle as in step 2.
4. Repeat for all cycles.



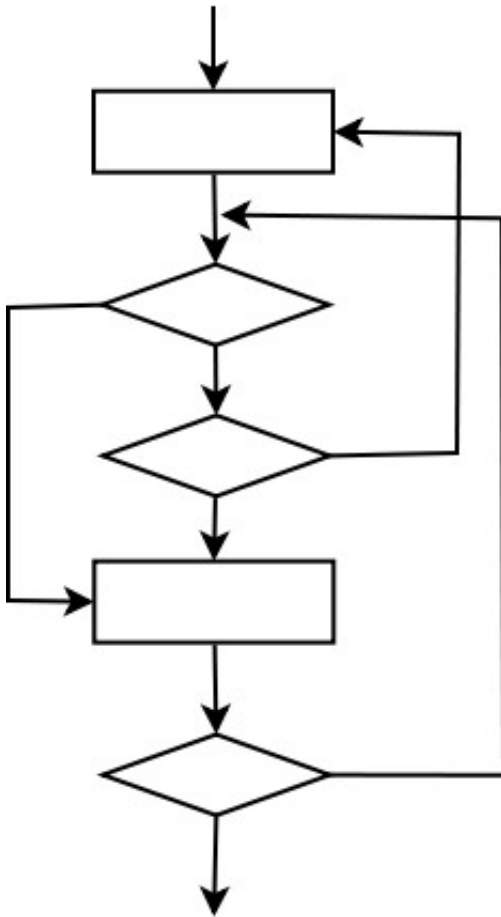
# Testing cycles

*Combined cycles*

Testing depends  
presence and / or absence of  
depending between cycles



# Testing cycles



*Unstructured  
cycles*