Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TOPIC 7: SOFTWARE TESTING

SOFTWARE TESTING

- Introduction
- Psycological aspects of software testing
- Information flow of software testing
- Indirect test.
- Direct test. Design of test cases
- Integration testing
- Debbuging

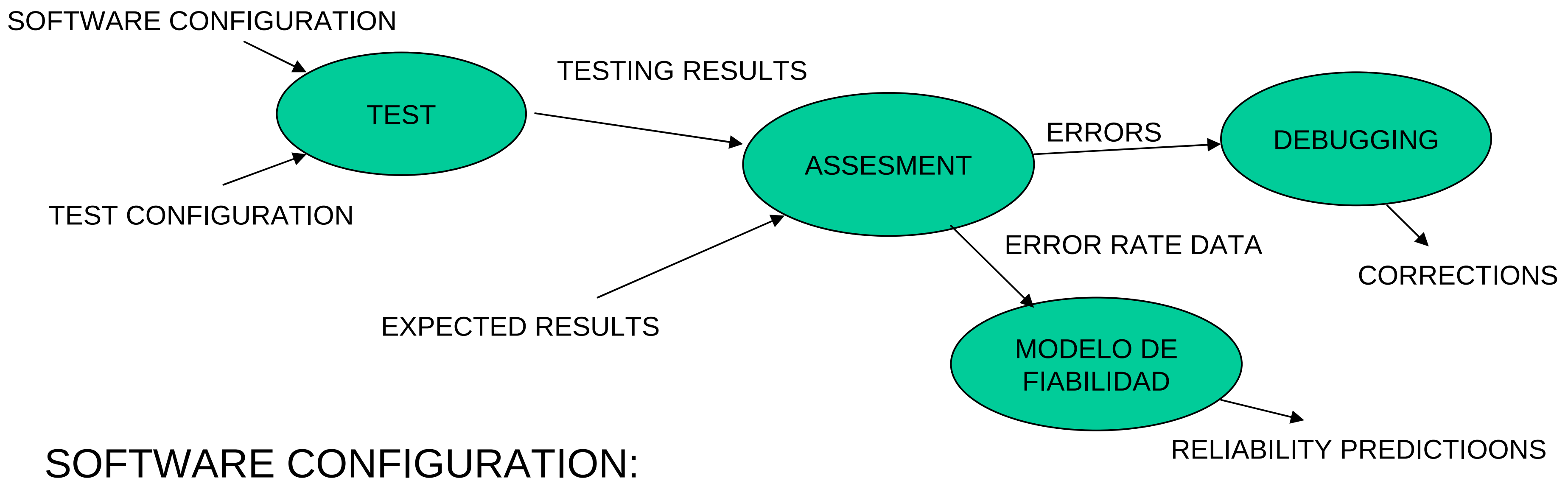
SOFTWARE TESTING TECNIQUES. INTRODUCTION

- It represents a final revision of the specification, design and encoding phases.
- Main goal of software testing:
  - To find errors
- The purpose of software testing is to design test data sets with a high probability to find a new error.

PSYCOLOGICAL ASPECTS OF SOFTWARE TESTING

- Software testing is usually not well regarded by programmers
- Some external aspects such as temporal limitations, product´s costs, product´s chances, … together the not well understood “destructive” nature of software testing does not make easy this task.
- In this sense, it might be justified to distrust of the programmer objectivity when he tests his appplication.
- That is the reason to involve an external referee team during the testing process.

FLOW OF INFORMATION



SOFTWARE CONFIGURATION:
- Requirement specification
- Design specification
- Source code

TEST CONFIGURATION:
- Process planning
- Test tools
- Test cases
- Expected results

INDIRECT TEST

- These methods are known as indirect since they are performed without a computer
- These ones are carried out just after the encoding phase and before testing the software using a computer.
- The indirect test involves a visual inspection of program code and they are performed by a team of analyst and programmers.
- Main goal: Find errors, no solutions.
- This testing approach detects groups of errors allowing after a massive correction.

DESIGNING TEST CASES

- There are two main product-testing approch:
  - **White-box tets:** Also known as clear box testing, glass box testing or structural testing uses an internal perspective of the system to design test cases based on internal structure.
  - **Black-box test:** takes an external perspective of the test object to derive test cases.The test designer selects valid and invalid input and determines the correct output. There is no knowledge of the test object's internal structure.

- A more complete software testing process is achieved combining both approachs.

DESIGN TECNIQUES TO DESIGN TEST CASES

Purpose:

- **Reduce** the number of test case mantaining the test effectivity**.**
- According to the approaches:
  - Black-box (what it does)
  - White-box (how it is done):

WHITE-BOX TEST

- As it was mentioned before these tecniques uses an internal perspective of the system to design test cases.
- The test cases generated by means of white-box approach have to ensure:
  - That all the independent paths of an specific module will be executed at least once.
  - That all the logical decisions will be performed taking into account both the te true and false values.
  - That all loops will be exexuted regarding their operational limits.
  - That all the internal data structure will be tested.

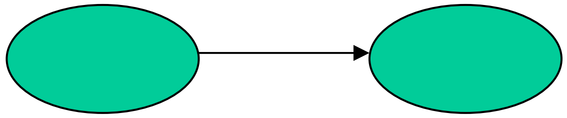
CICLOMATIC COMPLEXITY METRIC

- This metric allow us to measure the logical complexity of a procedural design.
- This measurement may be used to define a basic set of execution paths
- In order to determine the ciclomatic complexity a control flow graph will be used.

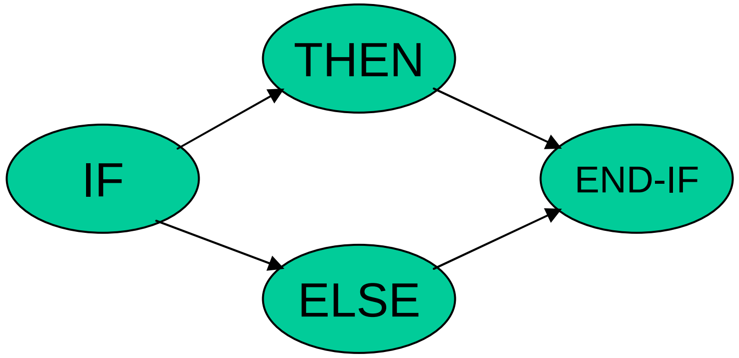
CONTROL FLOW GRAPH

- It is a representation, using graph notation, of all paths that might be traversed through a program during its execution.
- Each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps…
- Directed edges are used to represent jumps in the control flow.
- There are, in most presentations, two specially designated blocks: the *entry block*, through which control enters into the flow graph, and the *exit block*, through which all control flow leaves.

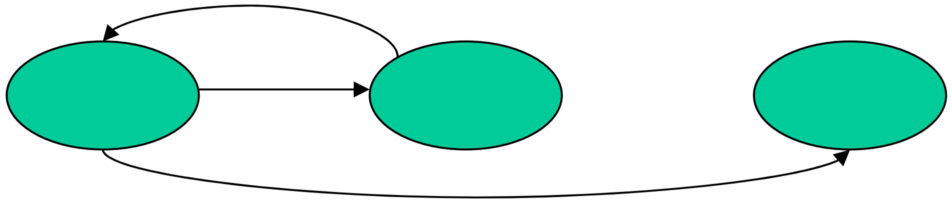
GRAPHICAL REPRESENTATION OF SOME COMMON CONTROL STRUCTURES



SEQUENCE

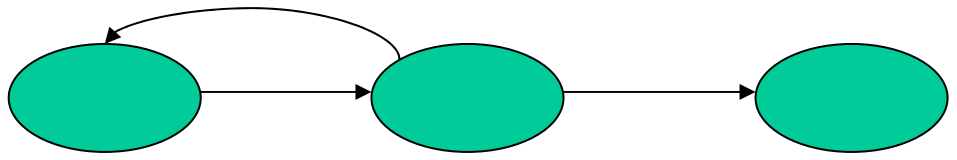


IF SELECTION



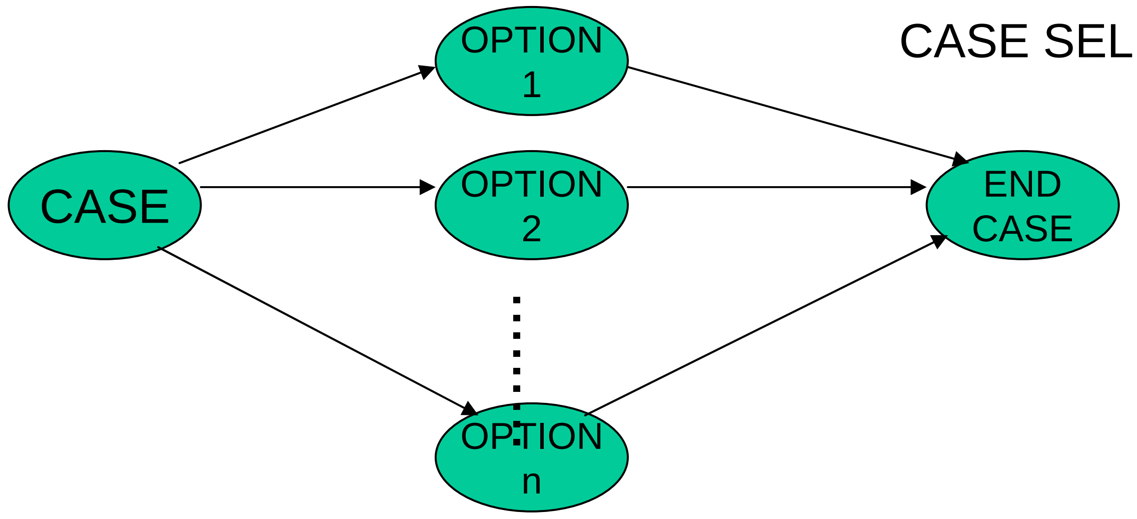
WHILE-LOOP

REPRESENTACIÓN DEL GRAFO DE FLUJO DE LAS ESTRUCTURAS DE CONTROL

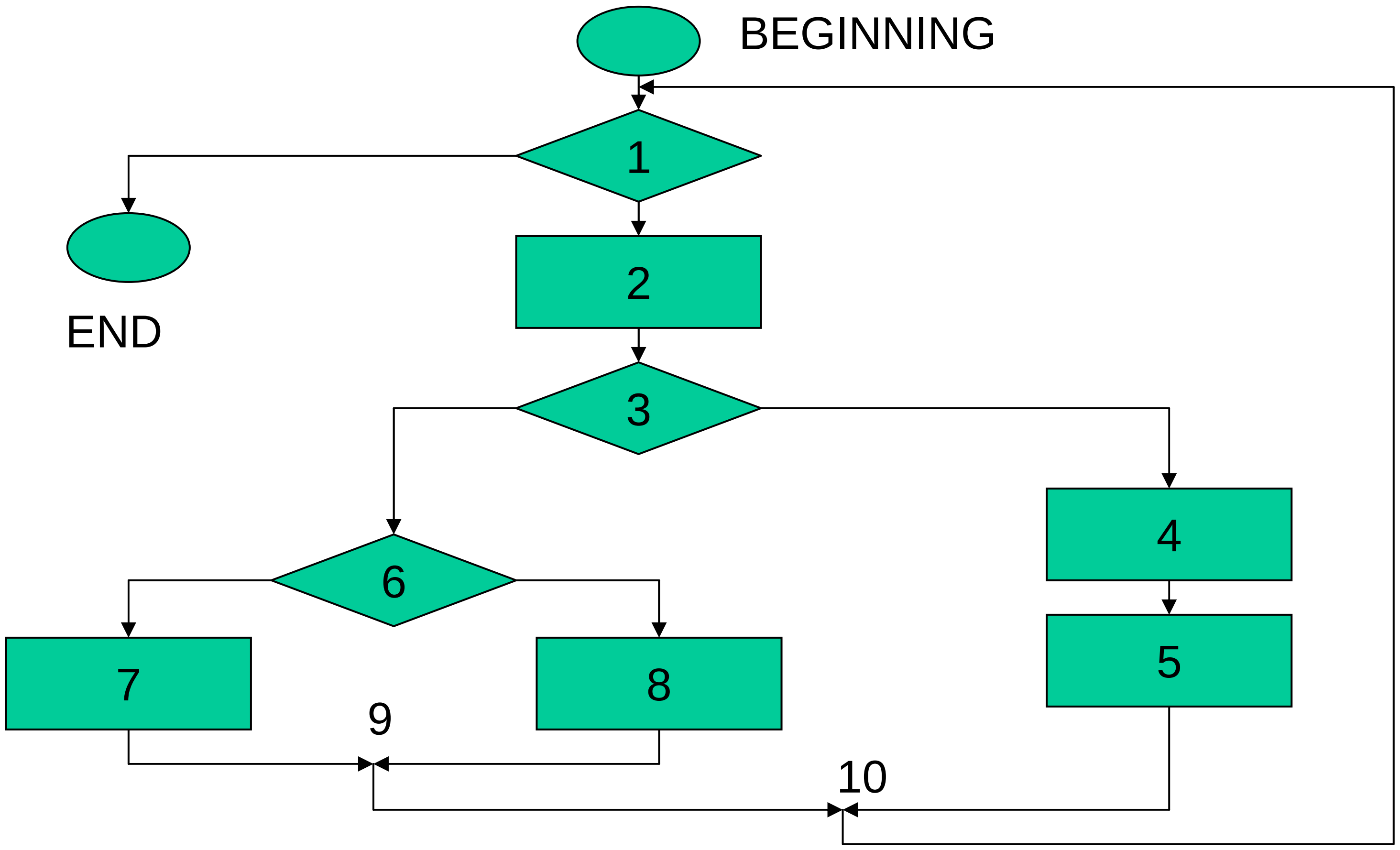


REPEAT-LOOP

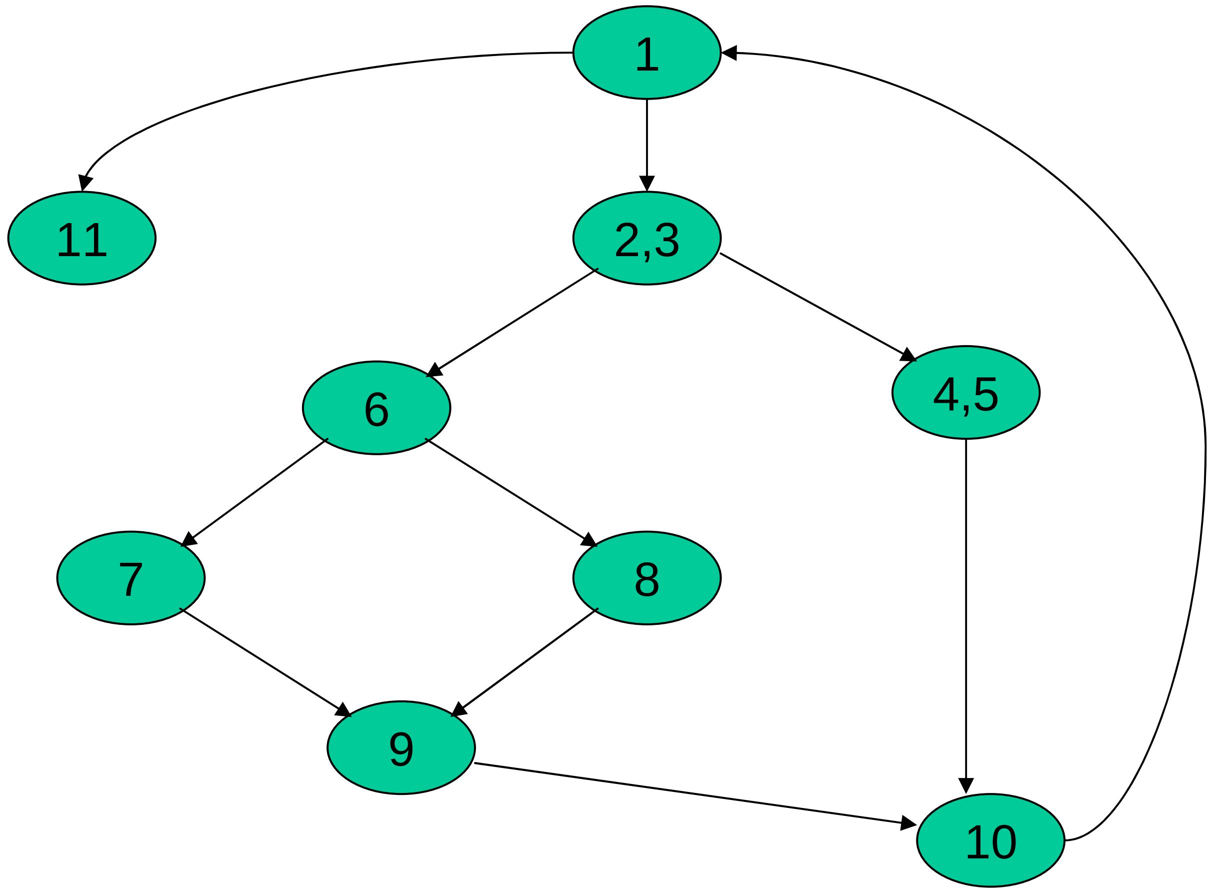
CASE SELECTION



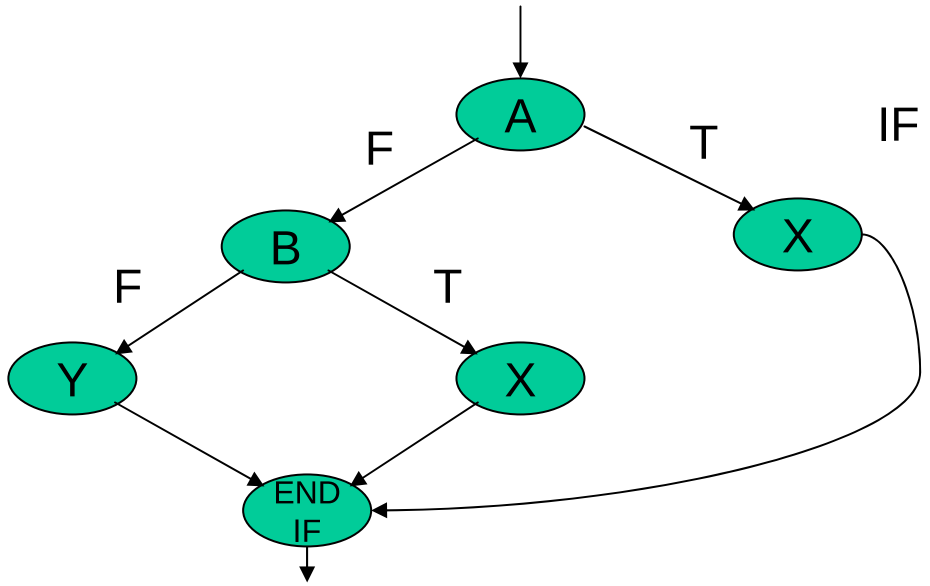
AN EXAMPLE DESCRIPTION USING A FLOWCHART



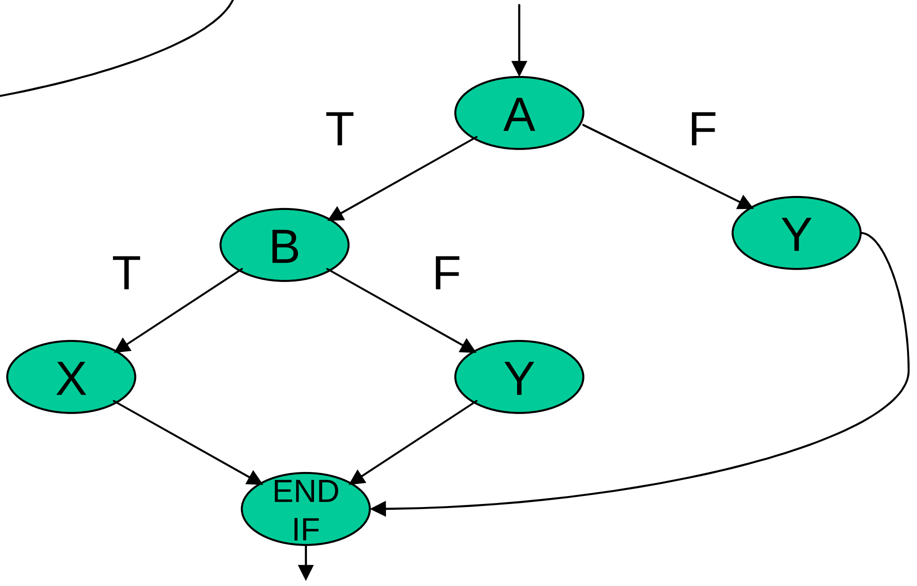
THE SAME EXAMPLE DESCRIPTION USING A CONTROL FLOW GRAPH.



FLOW GRAPHS RELATED TO LOGICAL COMPOSED CONDITIONS



```
IF A OR B
        THEN X
        ELSE Y
```



```
IF A  AND  B
        THEN X
        ELSE Y
```

CICLOMATIC COMPLEXITY

- It is a **software metric** to measure quantitatively the logical complexity of a program.
- Under this approach that represents to work out the maximum number of independent paths of a program.
- According to this context, an independent path is any logical path that introduces new statements. In terms of a graph that means a new direct edge that has not been considered yet.
- This quantitative mesure give us an **upper bound** to the number of test cases.

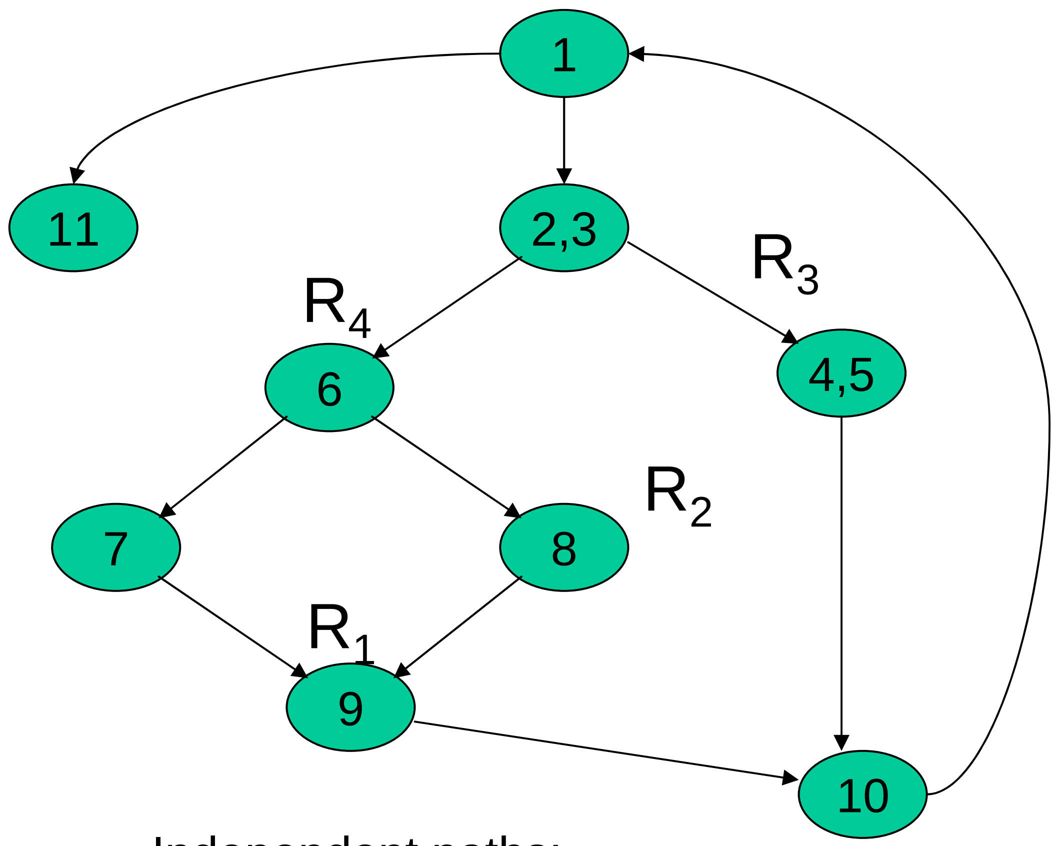
WORKING OUT THE CICLOMATIC COMPLEXITY

- Some ways to work out the ciclomatic complexity:
  - The ciclomatic complexity (V(G)) is equal to the numbers of region that compose the control flow graph.
  - V(G)=A-N+2
    - Where A is the number of direct edge and is the number of nodes that compose the control flow graph.
  - V(G)=P+1
    - Where P is the number of predicate nodes present on the control flow graph.

HOW TO OBTAIN THE TEST CASES

- The test cases are generated guaranting the execution of all the independent paths.

CICLOMATIC COMPLEXITY. INDEPENDENT PATHS



- V(G)=4 Regions
- V(G)= 11A-9N+2=4
- V(G)=3NP+1=4

Independent paths:

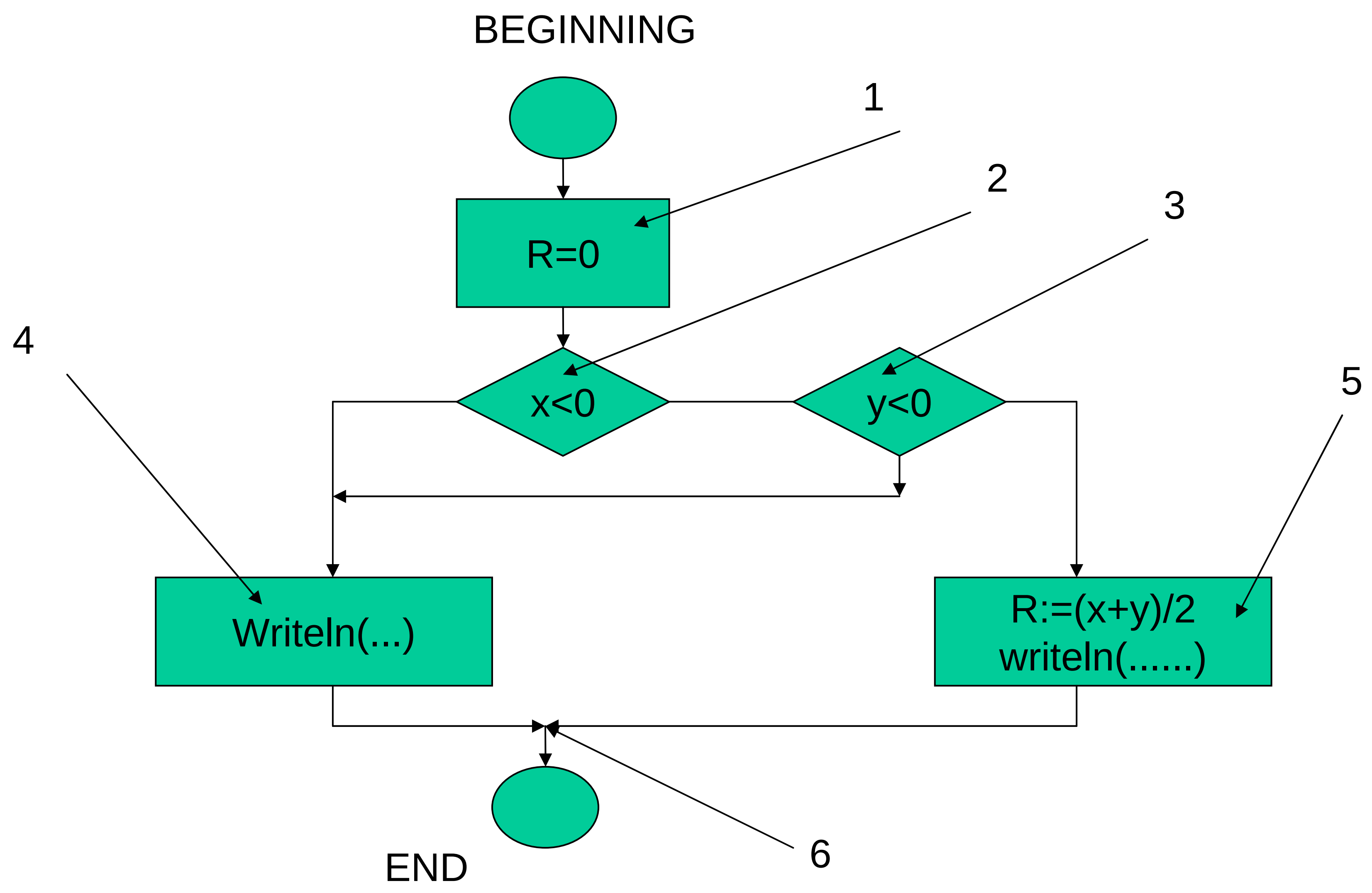
- Path 1: 1-11
- Path 2: 1-2-3-4-5-10-1-11
- Path 3: 1-2-3-6-8-9-10-1-11
- Path 4: 1-2-3-6-7-9-10-1-11

EXAMPLE

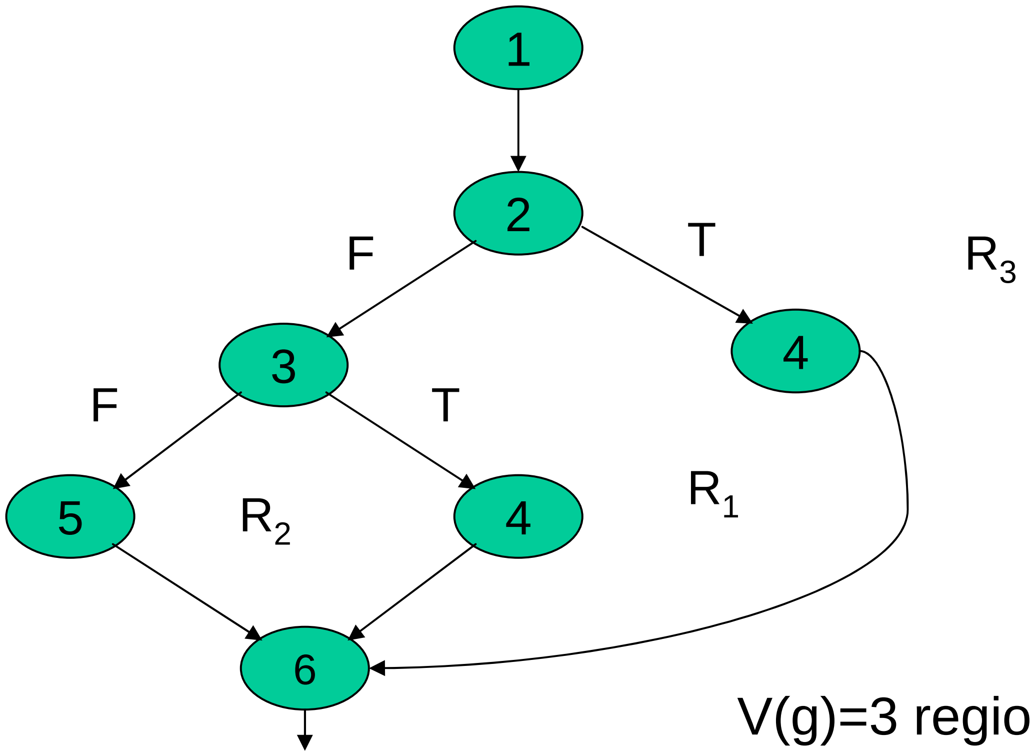
- Design test cases by means of ciclomatic complexity metric for the following code:

```
r:=0;
if (x<0 or y<0) then
        writeln(‘x e y must be a positive number’)
else begin
        r:=(x+y)/2;
        writeln(‘the average is: ‘, r)
        end;
```

EXAMPLE



EXAMPLE. CONTROL FLOW GRAPH



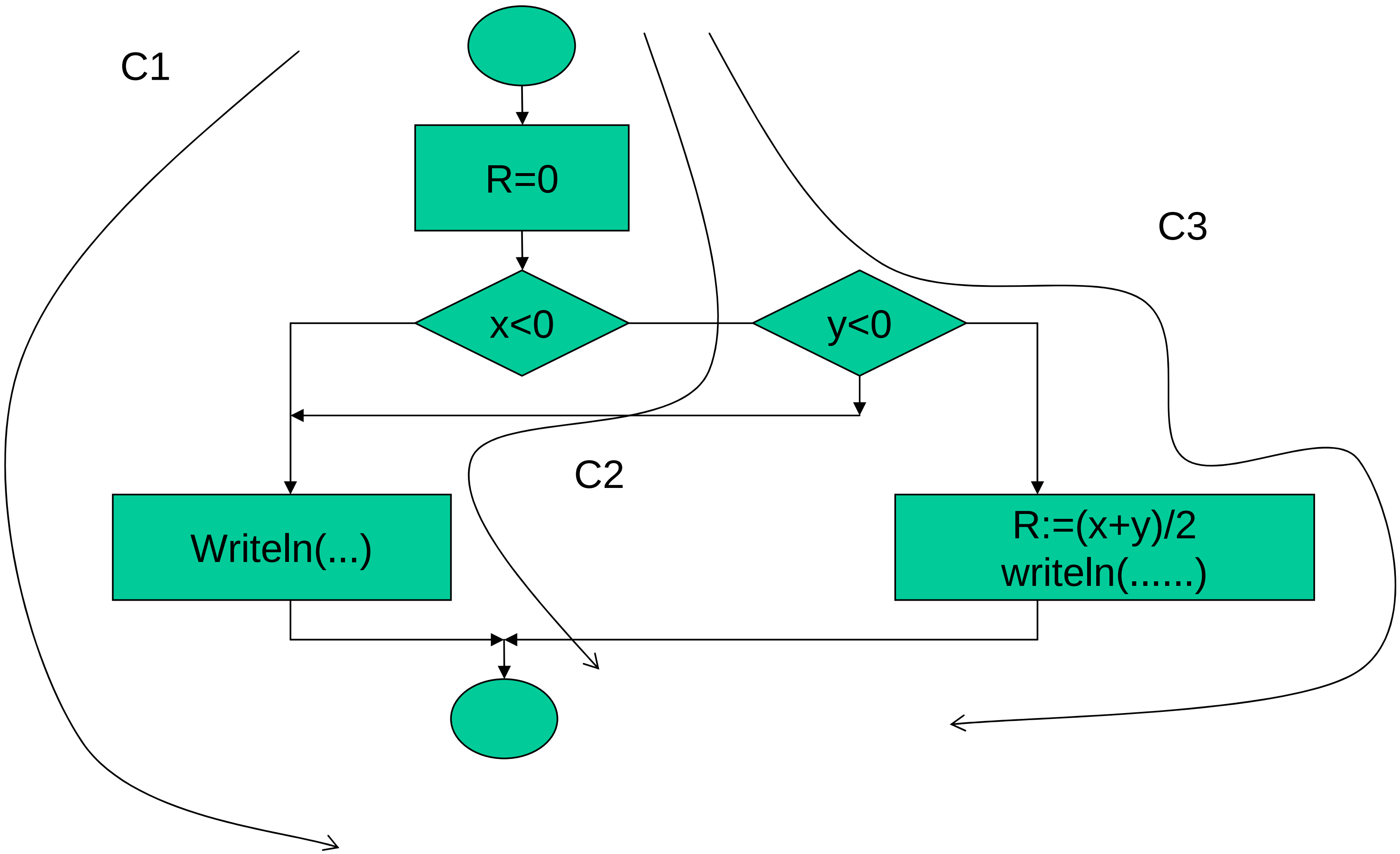
V(g)=3 regions=3
V(g)=8A-7N+2=3
V(g)=2NP+1=3

EXAMPLE. INDEPENDENT PATHS

- Since the ciclomatic complexity is three, this code has three independent paths:
  - C1: 1-2-4-6
  - C2: 1-2-3-4-5
  - C3: 1-2-3-5-6

- Therefore the test cases associated to these independent paths are:
  - C1: $x<0$ e (y irrelevant)
  - C2: $x\geq 0$ e $y<0$
  - C3: $x\geq 0$ e $y\geq 0$

EXAMPLE. INDEPENDENT PATHS



BLACK BOX APPROACH TO DESIGN TEST CASES

- This approach considers the product functionality.
- Therefore the testing process is performed over the system interface.
- This approach reduce the number of test cases by means of selecting valid and invalid inputs,

SOME KIND OF DETECTED ERRORS

- Absent or incorrect functions
- Interface errors
- Data base access errors
- Data structure errors
- Performance errors
- Initialization and completion errors.

BLACK BOX TECHNIQUES

- Equivalence partitioning
- Boundary value Analysis

EQUIVALENCE PARTITIONING

- It consists in dividing the input data field into a set of data classes known as equivalence classes.

EQUIVALENCE CLASS CONCEPT

- **An equivalence class** is a input set of data defining both valid and invalid states of the system.
  - Valid class: it generates an expected value.
  - Invalid class: it generates an unexpected value.
- This division may be carried out by means of the input conditions described in the specification.

INPUT CONDITIONS

- An input condition may be expressed as an statement of the specificacion.

An specific value	"..Enter **five** values.."
A range of values	"...the values are **between 0 and 10**..."
A related set of values	"....**Reserved words of a language**...."
A logic condition	"...**it must be**..."

THE PROCEDURE

- Identification of equivalence classes.
- Generation of test cases from the equivalence classes:
    - An equivalence class may be represented by any of their element.

IDENTIFICATION OF EQUIVALENCE CLASSES

- For each input condition a valid and invalid equivalence class may be identified.
- This procedure is heuristic.
- However there are some criteria to make easy the identification procedure.

EQUIVALENCE CLASS IDENTIFICATION CRITERIA

Input conditions	Valid equivalence classes	Invalid equivalence classes
1. Range of values	**1 EC** involving any value inside the range**.**	**2 EC** involving any value outside the range
2. Specific values	**1 EC** involving that specific value	**2 EC** involving two values, One above and one below wrt The specific value
3. A related set of values	**1 EC** for each element	**1 EC** representing one element outside the set of values
4. Logic Condition	**1 EC** satisfaying the logic condition	**1 EC** not satisfaying the logic condition

EQUIVALENCE CLASS TABLE

- The E. C. Table is the place where all the identified classes are described in order to find the different tets cases related to them.

Input conditions	Valid equivalence class	Invalid equivalence class

HOW TO OBTAIN THE TEST CASES FROM THE E.C.?

- Valid equivalence classes:
  - Each test case has to cover the maximum number of valid E.C.
  - it implies to find the minimum number of test cases in order to cover all the valid E.C..

- Generate a test case for each identified invalid equivalence class for avoiding hidden errors.

Example 1:

- Design a set of test cases to detect possible errors during the process of declaring an identifier taking into account the following syntactic rules of an hypothetic programming:
  - The identifier must be between 5 and 15 characters long.
  - These characters may be:
    - Upper and lower cases
    - Dígits (0,9)
    - Dash (-)
  - The system distiguishs between upper and lower cases.
  - Dash may not be placed neither at the beggining nor at the end of an identifier.
  - Several dashes may be placed consecutively inside the identifier
  - Any identifier must contain, at least, one alphabetic character.
  - The identifier may not be a reserved word of this programming language.

• Recording the input conditions

Input conditions	Valid Equivalence classes	Invalid equivalence classes
-Between 5 and 15 characters long		
-These charcaters may be upper and lower cases, digits and dashes.		
-The system distiguishs between upper and lower cases		
-Dash may not be placed neither at the beggining nor at the end of an identifier -Several dashes may be placed consecutively inside the identifier		
- Any identifier must contain, at least, one alphabetic character		
-The identifier may not be a reserved word of this programming language		

• Recording the equivalence classes

Input conditions	Valid equivalence classes	Invalid equivalence classes
-Between 5 and 15 characters long	**1**. 5≤nº characters. ≤15	**2**. nº characters<5
		3. 15< nº characters
- These charcaters may be upper and lower cases, digits and dashes.	**4.** Characters of the identifier ∈{letras, dígitos, guión}	**5.** Some characters of the identifier ∉{letras, dígitos, guión}
-The system distiguishs between upper and lower cases	**6.** Declared identifier ∈{Valid identifiers}	**7.** The same identifier but changing one character from upper to lower case or vicerversa
-Dash may not be placed neither at the beggining nor at the end of an identifier -Several dashes may be placed consecutively inside the identifier	**8.** Identifiers withour dashes at the ends of these ones and presenting several consecutive dashes inside them	**9**. Identifier presenting a dash at the beggining
		10. Identifier presenting a dash at the end
-Any identifier must contain, at least, one alphabetic character	**11.** At least one alphabetic character is present on the identifier	**12.** No alphabetic character is present on the identifier
- The identifier may not be a reserved word of this programming language	**13.** Identifier ∉{reserved words}	**14, 15, 16** ...One case for each reserved word.

- Obtaining the test cases

Identificador	Clases de equivalencia cubiertas	Resultado
Num-1---d3 (10)	1,4,6,8,11,13 (regarding all the valid Equivalence classes)	Identifier accepted by the system
Nd3	2	Error message
Num-1-letra3---d3 (17)	3	Error message
Nu%m-1---d3 (11)	5	Error message
NuM-1---d3 (10)	7	Error message
-um-1---d3 (10)	9	Error message
Num-1---d- (10)	10	Error message
456-1---23 (10)	12	Error Message
Real	14	Error message
..(The remaining reserved words)..	15,16......	Error message

THE BOUNDARY VALUE ANALYSIS (BVA)

- BVA is based on the experimental evidency that errors are usually localized on the <u>**boundaries of input data fields**</u>.

- That kind of considerations enhances the test eficiency.
    - <u>**Boundary conditions**</u> : these conditions represent values just above and below of one equivalence class.

GENERATING THE TEST CASES

- It supposes to generate as many test cases as it is necessary for considering all the boundary conditions.
- There are, as Equivalence partitioning, some heuristic criteria to facilitate the identification process

Input and output conditions	Defining the test cases
1. Range of values as an input condition	**1** case to consider the maximum value of the range
	1 case to consider the minimum value of the range
	1 case to consider just the value above the maximum
	1 case to consider just the valuie below the minimum
2. An specific numeric value as an input condition.	**1** case to consider the specific numeric value
	1 case to consider just the value above the specific value
	1 case to consider just the value below the specific value
3. Range of values as an output condition	As in the item 1
4. An specific numeric value as an output condition.	As in the item 2
5. A data structure as an output or input condition.	**1** case to consider the first element of the data structure
	1 case to consider the last element of the data structure

Example 2:

- Let us suppose a program that works out, given the sides , what kind of triangle it is.
- Triangle conditions:

  $A+B>C$ **and** $A+C>B$ **and** $B+C>A$

  - According to the E.P. we should consider just one valid and invalid class.

    {A=4, B=5, C=3}, {A=1, B=2, C= 5}

  - This approach would not detect an error as $A+B\geq C$

  - However BVA includes the case {A=1, B=2, C=3}

- BVA applied to example 1

Condition	Descriptions of test cases
Between 5 y 15 characters	**1** case involving 15 charcaters
	1 case involving 5 characters
	1 case involving 16 characters
	1 case involving 4 characters

Condition	Test cases
Between 5 y 15 characters	Num-1-let-3---d3 (15)
	Numd3 (5)
	Num-1-letr-3---d3 (16)
	Nud3 (4)

SUMMARIZING

- The testing process consist in **executing the program for detecting errors**.
- The testing process is not exhaustive.
- **The goal**: reduce the number of test cases keeping up the efficiency.
- There are two approaches:
  - **Black box**: take into account the functionality of the system.
  - **White box**: take into account the internal logic

SUMMARIZING

- White box test case design techniques:
  - **Ciclomatic complexity:**It is a **software metric** to measure quantitatively the logical complexity of a program.
- Black box test case design techniques:
  - **Equivalence partitioning**: it divides the input data field into valid and invalid equivalence classes.
  - **Boundary value analysis**: it consists in exercising the boundary condition of all the equivalence classes.
- A more effective black box testing process involves both techniques.