Department of Computer Science Academic Year 2024-2025

Software Testing and Quality Analysis

Module Supervisor: Ms. M. Halassa

Chapter 5:
Functional Testing Methods

Contents

- Introduction
- Functional testing
- Black box testing techniques
  - Partitional analysis.
  - Boundary value analysis
  - Cause-effect graphs.
- Conclusion

Introduction

- **Tests are** used to **validate** an application or module throughout its development. They are very often carried out on two levels:

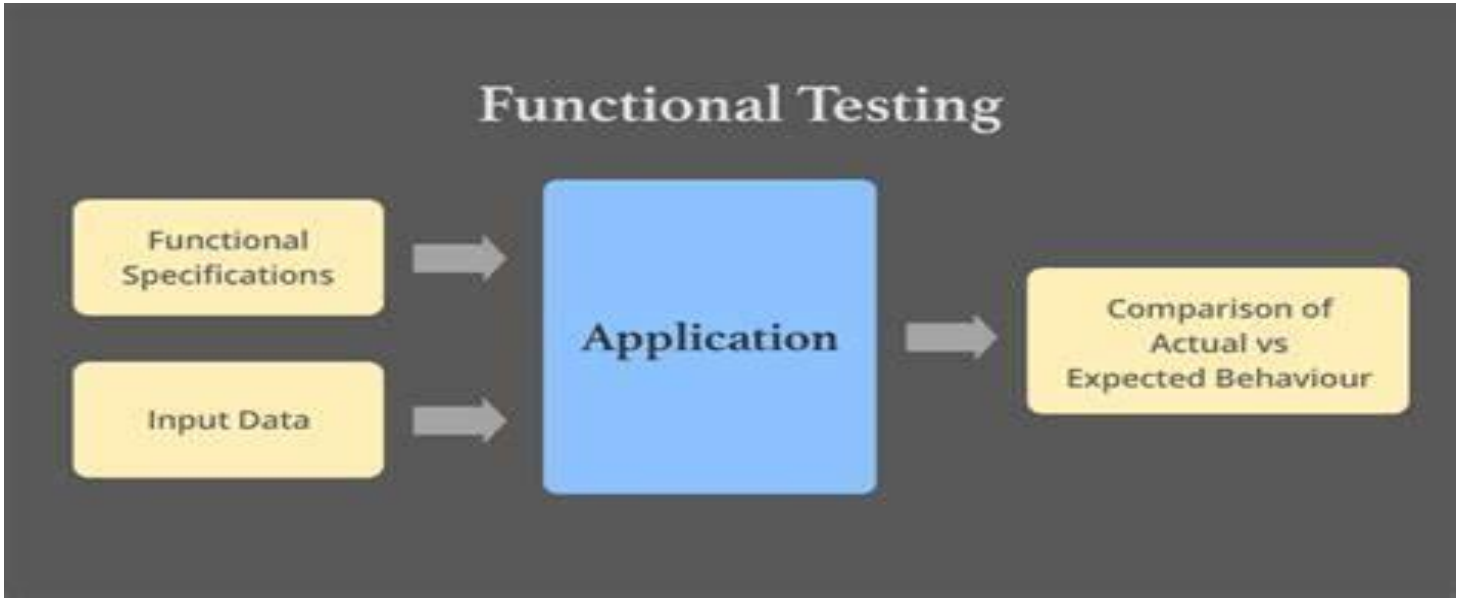
✓ Structural level: That is to say at the code level.

✓ Functional level: Represented by tests carried out on low-level functionalities, such as high levels.

- To start using functional testing, you need to familiarize yourself with its basic concepts.

Functional testing: Black-box testing

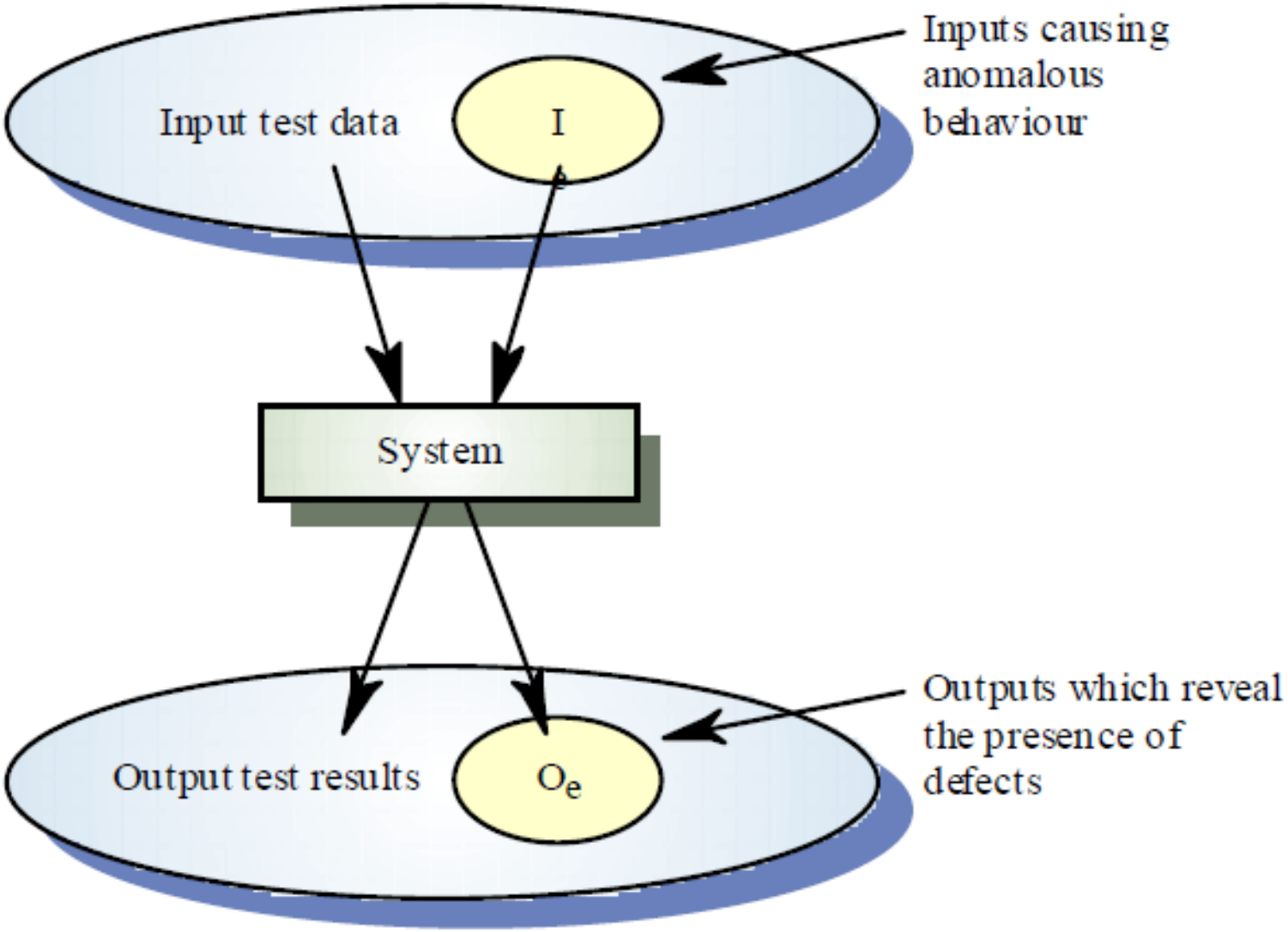
- Black box testing (also called functional testing) is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.



Functional test methods

- Here's how to create a functional test:
  - From the specifications we can predict **the test results**.
  - Always from the specifications we can define **the test data**.
  - By passing this test data into the program to be tested, we can compare the **execution results** with **the test results.**

Functional test methods



Software specification:

- A specification must describe at a minimum:

– the functions to be performed by the software,
– the interfaces of this software
– the constraints imposed on the developer.

Black box testing techniques

- Partitional analysis/Equivalence Class Testing
- Boundary value analysis
- Cause-effect graphs.

1-Partitional analysis(equivalence classes)

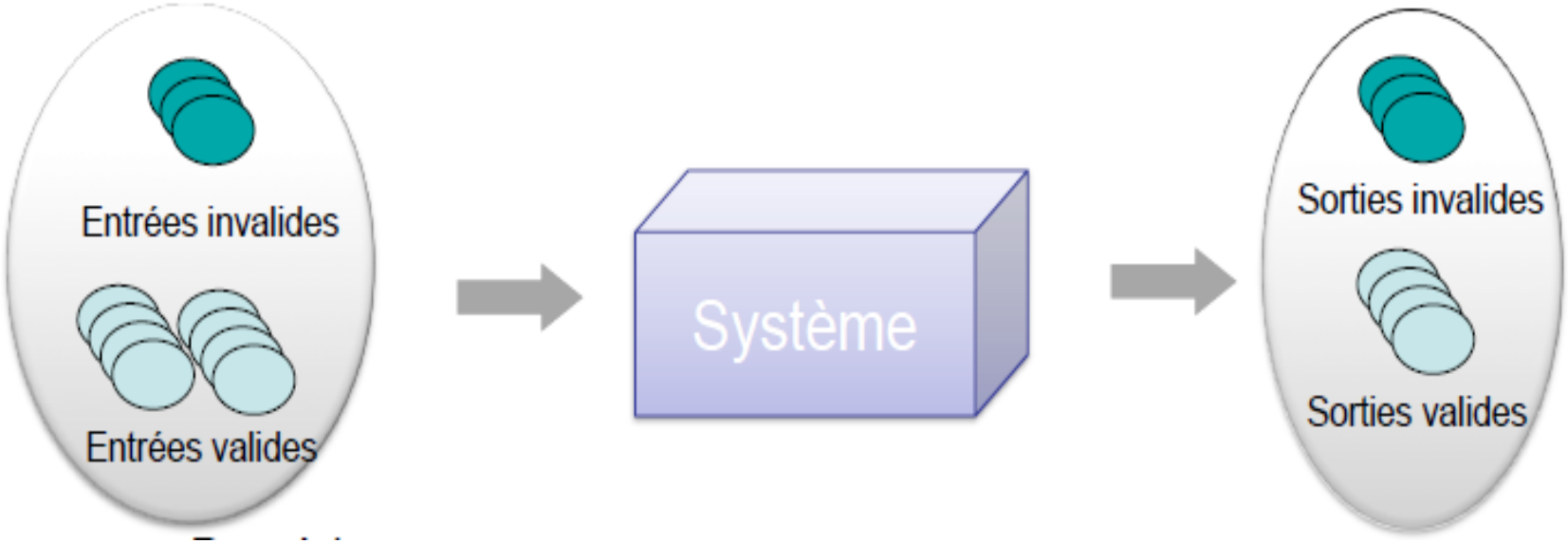
- The method **of partition analysis** based on the specifications of a program establishes equivalent classes (partitions).

- **Principle**: divide the domain of entries into a finite number of classes such that the program reacts in the same way for all the values of a class.

- **Consequence:** only one value per class should be tested!
- Avoid redundant testing .

Partitional analysis

- An ***equivalence class*** corresponds to a set of test data supposed to test the same behavior, i.e. activate the same fault.



Partitional analysis

- **Domain Partitioning Rules**

1 If the value belongs to an interval, construct:

- a class for lower values,
- a class for higher values,
- n valid classes.

2 If the data is a set of values, construct:

- a class with the set empty,
- a class with too many values,
- n valid classes.

3 If the data is an obligation or a constraint (form, meaning, syntax), construct:

- a class with the constraint respected,
- a class with the disregarded constraint

Partitional analysis: example1

- Suppose that the data at the input is a natural number that must contain five digits, therefore a number between 10,000 and 99,999.

- Partitioning into equivalence classes would then identify the following **three classes** (three possible conditions for input):

a. N < 10000

b. 10000 <= N <= 99999

c. N > 99999

Partitional analysis: example1

- We will then choose **test cases representative of** each class, for example, in the middle and at the borders (borderline cases) of each of the classes:

a. 0, 5000, 9999

b. 10000, 50000, 99999

c. 100000, 100001, 200000

Partitional analysis: example2

- Either a program calculating the absolute value of a relative integer entered in the form of a string of characters on the keyboard, what equivalence classes to test it?

Partitional analysis: example2

- **Class 1**

Invalid input, empty string

- **Class 2**

Invalid input, string with multiple words « 123 983 321»

- **Class3**

Invalid input, a single word but not a relative integer « 123.az+23 »

- **Class 4**

Valid input, a word representing a positive or null relative integer « 673.24»

- **Class 5**

Valid input, a word representing a negative relative integer « -2.4 »

Partitional analysis: example3

- The program to be specified receives as input three integer values supposed to represent the lengths of the sides of a triangle. He must check whether these values can define a triangle.
- If the answer is positive, the program must calculate the type of triangle:
  - scalene, isosceles, equilateral.

- Integer values x, y and z can define a triangle if and only if they satisfy the following inequalities:
  - $x < y + z \;\&\&\; y < z + x \;\&\&\; z < x + y$

Partitional analysis: example3

- Which test cases for this program with equivalence classes?
- ***Not a triangle***: 1 side at a length greater than the sum of the other 2
- ***Isosceles:*** 2 equal sides only
- ***Equilateral***: 3 equal sides
- ***Scalene:*** The others

2- Boundary value analysis

- Most programmers have probably found that, often, the errors are due to the fact that they did not predict the behavior of a program for limit values *(very high values, zero values of an index, negative values)*.
- Boundary testing covers a wide range of errors, as boundaries form the preferred natural host for most defects .

2- Boundary value analysis

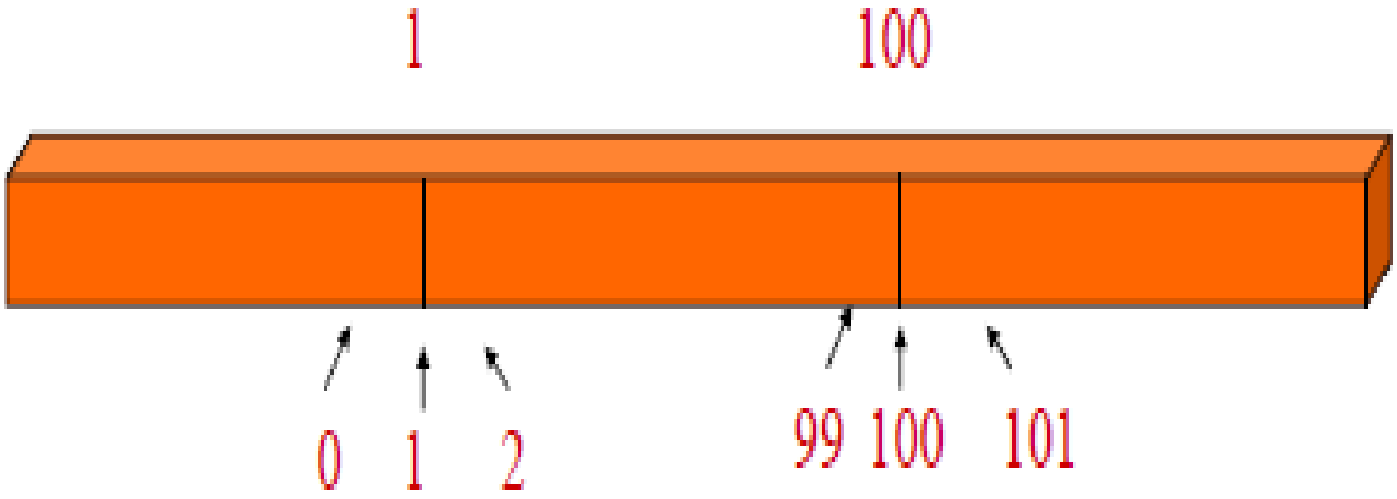
❖Principle: we are **interested in the limits of the intervals** partitioning the domains of the input variables

- For each interval, the 2 values corresponding to the 2 limits are kept, and the 4 values corresponding to the values of the limits ± the smallest possible delta

  n ∈3 .. 15 ⇒v1 = 3, v2 = 15, v3 = 2, v4 = 4, v5 = 14, v6 = 16

- If the variable belongs to an ordered set of values, we choose the first, second, penultimate and last n ∈ {-7, 2, 3, 157, 200} ⇒ v1 = -7, v2 = 2, v3 = 157, v4 = 200

Test at the limits of the interval [1,100]



3- Cause and effect graphs

- The cause-effect graph method can be seen as a separate method or as a method for creating relational partitions in the multi-variable case
- **Additional gain:** reflection on specifications, finding inconsistencies or omissions.

Steps

- make the graph .
- simplify / reduce the graph (optional) .
- Generating the decision table
- Produce test cases after simplification of decision matrix.

Cause and effect graphs

- Improvement of the equivalence class method
- To analyze input-output combinations
- Applicable on a subset

Cause and effect graphs

- Causes **C** are possible inputs (input variable, user actions, environment values, etc.)
- The **E** effects are the outputs of the system .
- The method consists of identifying the causes and effects of the system from the specifications, then translating the logic of the specification into a graph linking ***causes and effects.***
- ***nodes:*** causes (roots), effects (leaves), connectors or (intermediate)
- ***arcs:*** normal or negation.

Cause and effect graphs

- Identify and number entry conditions, causes
- Identify and number the output effects.
- The construction of the network is based on four types of usual symbols expressing the interdependencies of effects to causes:

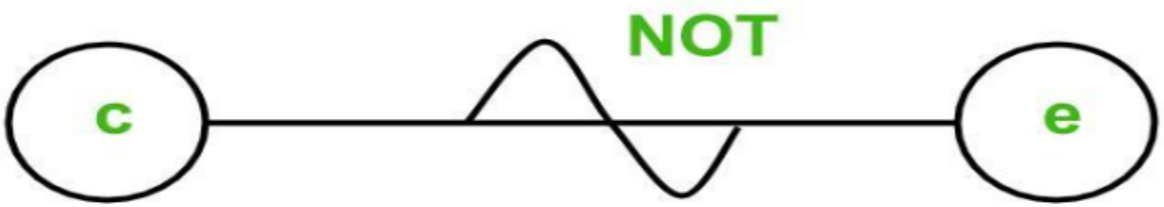
(identity, negation, logical OR and logical and)

Basic Notations used in Cause-effect graph:

- Here **c** represents **cause** and **e** represents **effect**

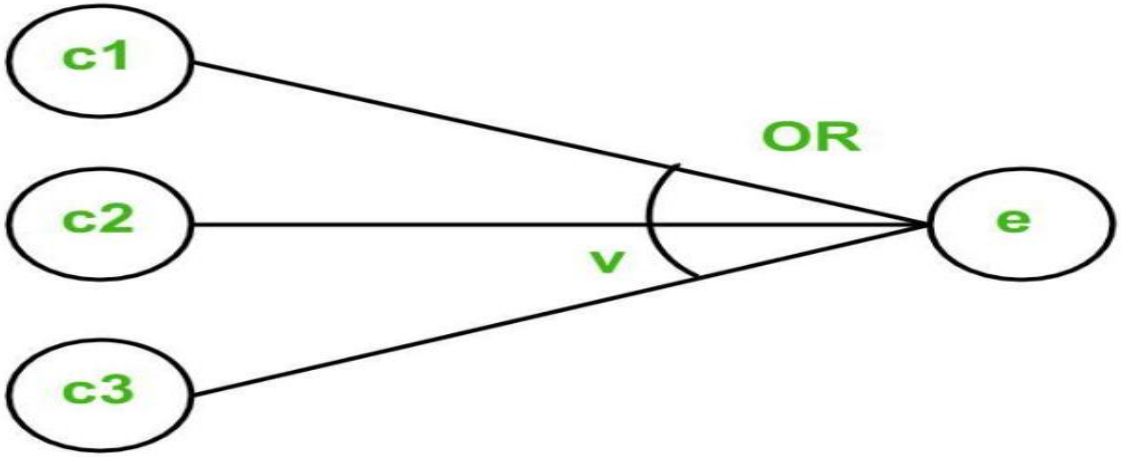
1. Identity Function: if c is 1, then e is 1. Else e is 0

2. NOT Function: if c is 1, then e is 0. Else e is 1.

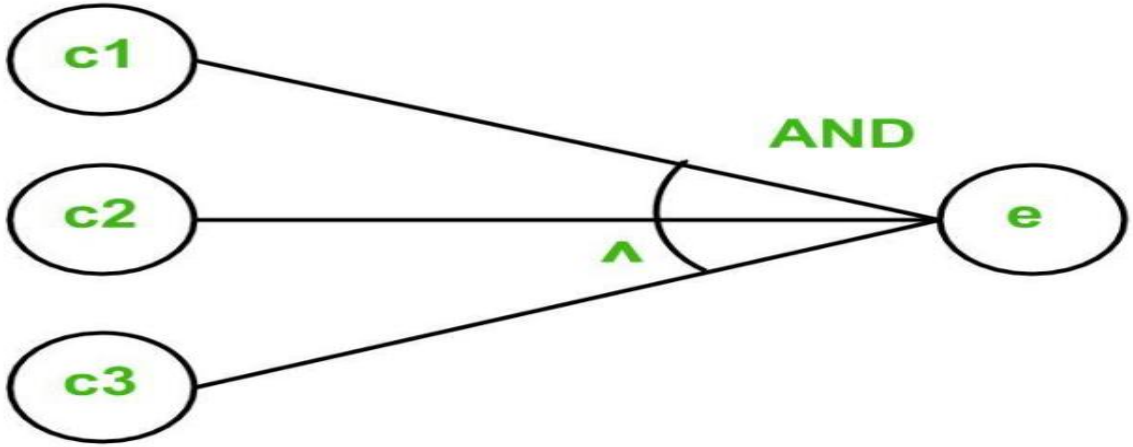


Basic Notations used in Cause-effect graph:

3. OR Function: if c1 or c2 or c3 is 1, then e is 1. Else e is 0



4. AND Function: if both c1 and c2 and c3 is 1, then e is 1. Else e is 0.



Cause-effect graphs: example

- "The character in column 1 must be an A or B.

In column 2, there must be a digit. If the first character is wrong, the X12 error message should be printed. If in column 2 we do not have a digit, then the message X13 is printed "

- **Analysis of causes**

A1: the character in the first column is A

A2: the character in the first column is B

A3: the character in the second column is a digit

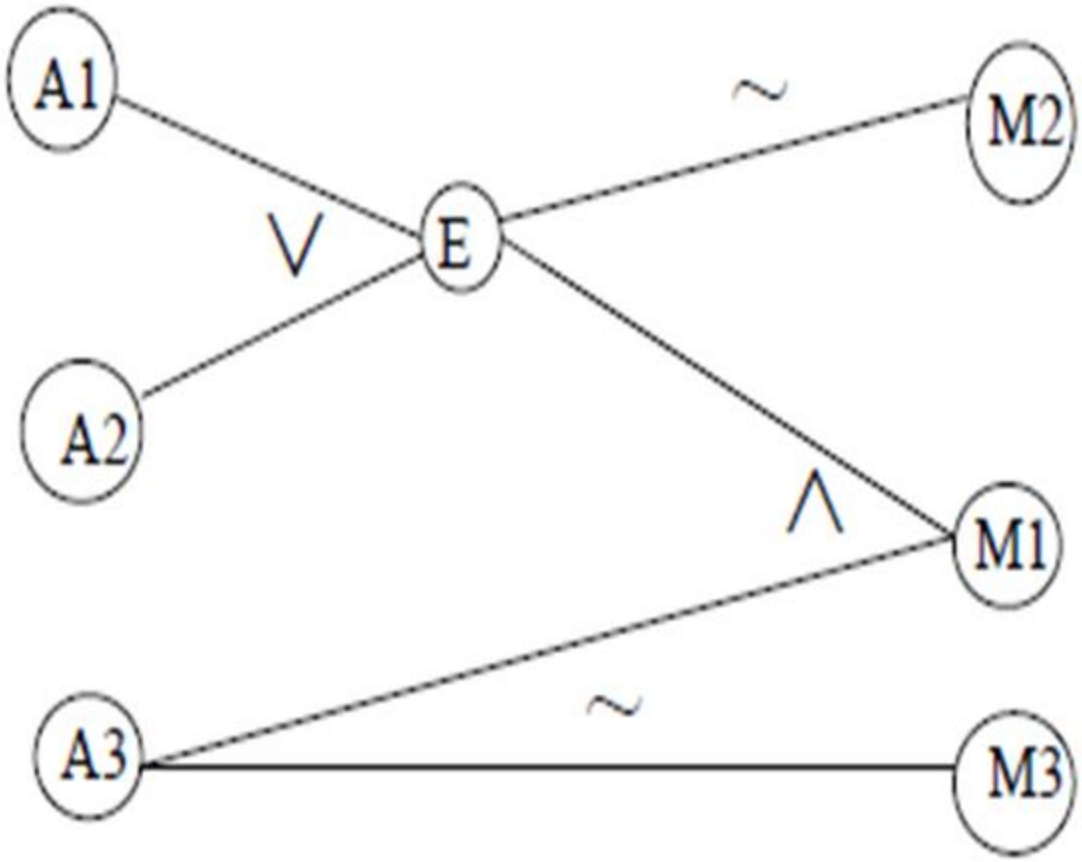
- **Analysis of effects** (resultats)

M1: program ok

M2: X12 message

M3: X13 message

Cause-effect graphs: example



- Analysis of causes

A1: char in col 1 is A

A2: char in col1is B

A3: char col 2 is a digit

-Analysis of effects

M1: program ok

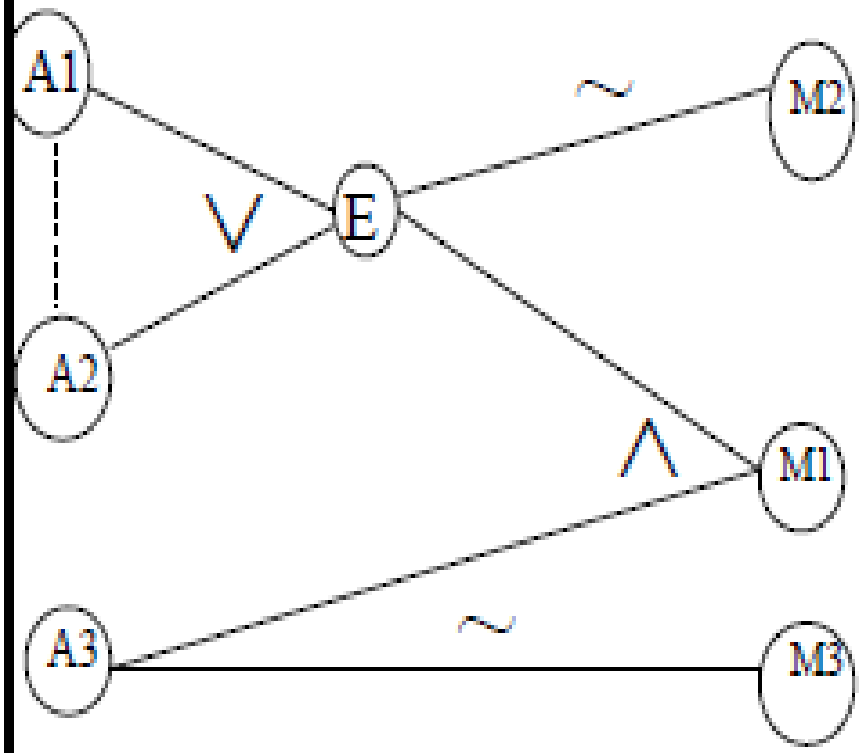
M2: X12 message

M3: X13 message-

Cause-effect graphs: example

- Causes and effects are seen as boolean variables
- Automatic generation of a decision matrix (Boolean matrix, True: 1 False: 0) from the cause-effect graph
- Elimination of invalid combinations: "unobservable behaviours" **e.g.** true A1 and A2,

Cause-effect graphs: example



Cause -effet	A1	A2	A3	M1	M2	M3
CT 1	0	0	0	0	1	1
CT 2	0	0	1	0	1	0
CT 3	0	1	0	0	0	1
CT 4	0	1	1	1	0	0
CT 5	1	0	0	0	0	1
CT 6	1	0	1	1	0	0

Generation of test cases

	A1	A2	A3	M1	M2	M3
CT 1	0	0	0	0	1	1
CT 2	0	0	1	0	1	0
CT 3	0	1	0	0	0	1
CT 4	0	1	1	1	0	0
CT 5	1	0	0	0	0	1
CT 6	1	0	1	1	0	0

→ « Ca », X12, X13

→ « C1 », X12

→ « Bz », X13

→ « B3 », ok

→ « Ak », X13

→ « A4 », ok

-Analysis of causes

A1: char in col 1 is A

A2: char in col1 is B

A3: char col 2 is a digit

-Analysis of effects

M1: program ok

M2: X12 message

M3: X13 message-

Reduction of the decision matrix

- If n=number of causes, m= number of effects

Then the decision matrix size is : $2^n \times (n+m)$

How we simplify the decision matrix?

Idea: Eliminate redundant TCs by identifying those with similar behaviors (same effects) even though they have different causes.

Example: TC3 and TC5

Conclusion

- Functional tests and structural tests are complementary and make it possible to highlight different defects in a program. Functional tests mainly detect defects due to a misunderstanding of the software specifications.
- Structural tests make it possible to detect faults related to programming.