CUI Abbottabad

Department of Computer Science

SOFTWARE TESTING

Lecture 3 & 4

Black Box Test Case Design Techniques

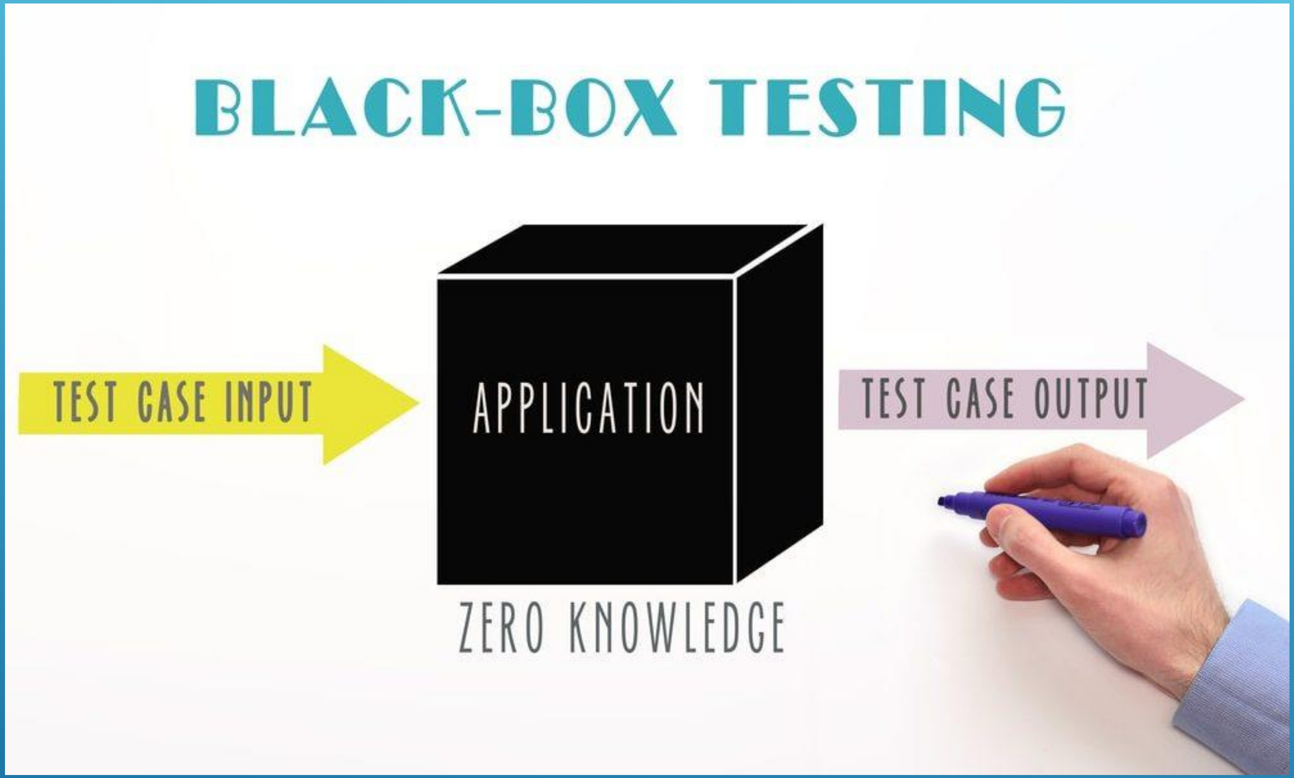
BLACK BOX TESTING

- Method that examines the functionality of an application, without looking at its structure.
- The tester does not ever examine the programming code and does not need any further knowledge of the program other than requirement specifications (SRS).
- You can begin planning for black box testing soon after the requirements and the functional specifications are available.
  - A benefit of a specification-based approach is that the tests are looking at what the software should do, rather than how it does.
- Expected outcomes can be generated if they are stored in the specification, assuming that the stored specification is actually correct.

BLACK BOX TESTING

- The designer and the tester are independent of each other.
- The testing is done from the user point of view.
- Test cases can be designed after requirements are clear.
- A good test case is one that has a reasonable probability of finding an error.
- Exhaustive-input testing of a program is impossible.
  - Limited to trying a small subset of all possible inputs.
- Select the right subset-> the subset with the highest probability of finding the most errors.

BLACK BOX TESTING

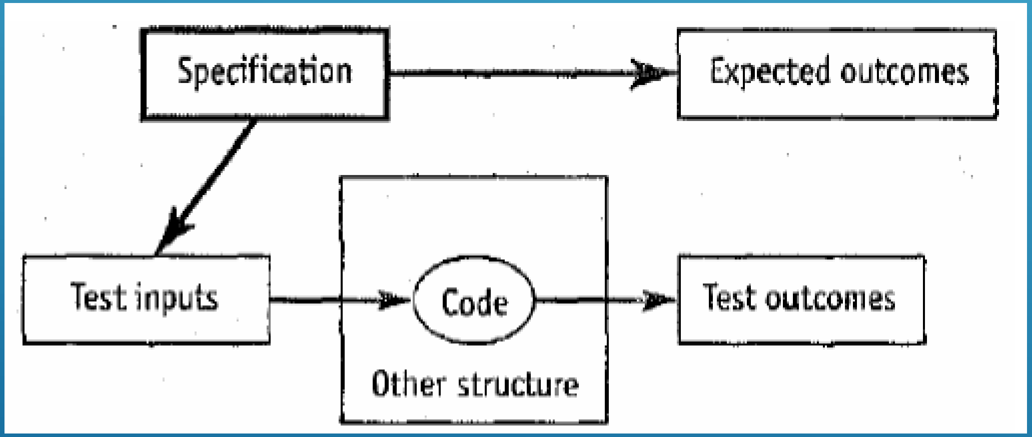


TYPES OF BLACK BOX TESTING

- **Functional testing –** This black box testing type is related to the functional requirements of a system; it is done by software testers. Functional testing is concerned only with the functional requirements of a system and covers how well the system executes its functions.
- **Non-functional testing –** Non functional testing is concerned with the non-functional requirements and is designed specifically to evaluate the readiness of a system according to the various criteria (non-functional requirements such as performance, scalability, usability etc.) which are not covered by functional testing.
- **Regression testing –** Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

TEST DATA

- For Test Case We First Need Test Data
- We need to have a set of thought processes that let us select test data more intelligently.
- Exhaustive black-box and white-box testing are, in general, impossible, but suggested that a reasonable testing strategy might be elements of both.

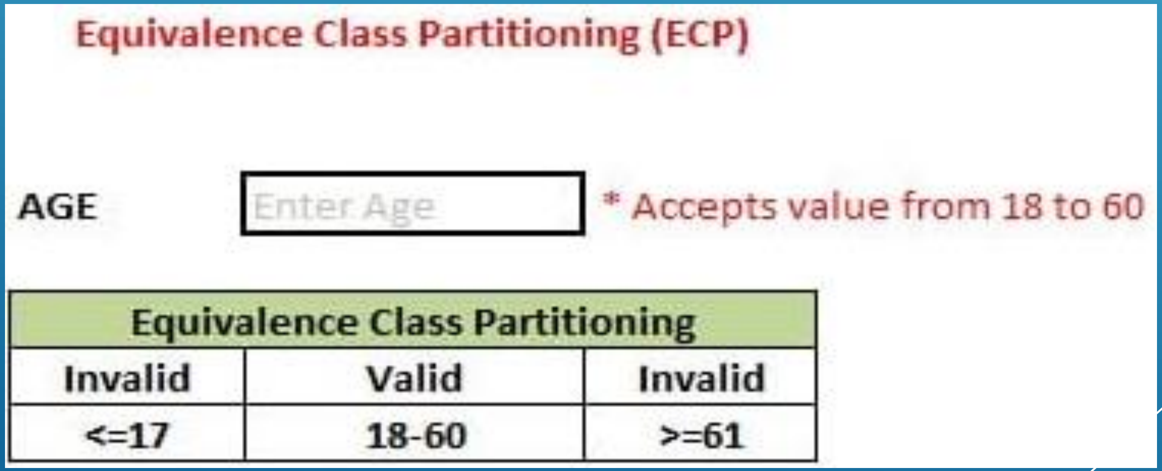


BLACK BOX TEST CASE DESIGN TECHNIQUES

- ❑ Equivalence Partitioning
- ❑ Boundary Value Analysis
- ❑ Error Guessing
- ❑ Cause Effect Graph

EQUIVALENCE PARTITIONING

- Dividing the test input data into a finite number of equivalence classes (range of values) and selecting one input value from each range is called **Equivalence Partitioning**.
- This technique is used to reduce an infinite number of test cases to a finite number.
  - while ensuring that the selected test cases are still effective test cases which will cover all possible scenarios.
  - If one test case in an equivalence class detects an error, all other test cases in the equivalence class would be expected to find the same error.



Equivalence Class Partitioning		
Invalid	Valid	Invalid
<=17	18-60	>=61

EQUIVALENCE PARTITIONING

- If one application is accepting input range from 1 to 100, using equivalence class we can divide inputs into the classes, for example, one for valid input and another for invalid input and design one test case from each class.

- In this example test cases are chosen as below:

- One is for valid input class i.e. selects any value from input between ranges 1 to 100. So here we are not writing hundreds of test cases for each value. Any one value from this equivalence class should give you the same result.
- One is for invalid data below lower limit i.e. any value below 1.
- One is for invalid data above upper limit i.e. any value above 100.

EQUIVALENCE PARTITIONING EXAMPLE (1/3)

- Employees of an organization are allowed to get accommodation expenses while traveling on official tours. The program for validating expenses claims for accommodation has the following requirements
- There is an upper limit of Rs. 3,000 for accommodation expense claims
- Any claim above Rs. 3,000 should be rejected and cause an error message to be displayed
- All expense amount should be greater than zero and an error message to be displayed if this is not the case

TEST CASE GENERATION (USING BLACK BOX) AND TEST DATA GENERATION (EQUIVALENCE CLASS PARTITIONING) EXAMPLE (2/3)

- **Inputs:** Accommodation Expense

- **Partition the Input Values (ECP):**

Valid:

$0 < \text{Expense} \leq 3{,}000$

Invalid:

Expense ≤ 0
Expense > 3,000
Special Character
Alphabets

1.First we made Test Data

TEST CASE GENERATION (USING BLACK BOX) AND TEST DATA GENERATION (EQUIVALENCE CLASS PARTITIONING) EXAMPLE (3/3)

Test Case ID	1	2	3
Expenses	2000	-10	3500
P. tested	0<Expense<3000	Expense<0	Expense>3000
Exp-output	OK	error message	error message
Act-output	OK	error message	error message

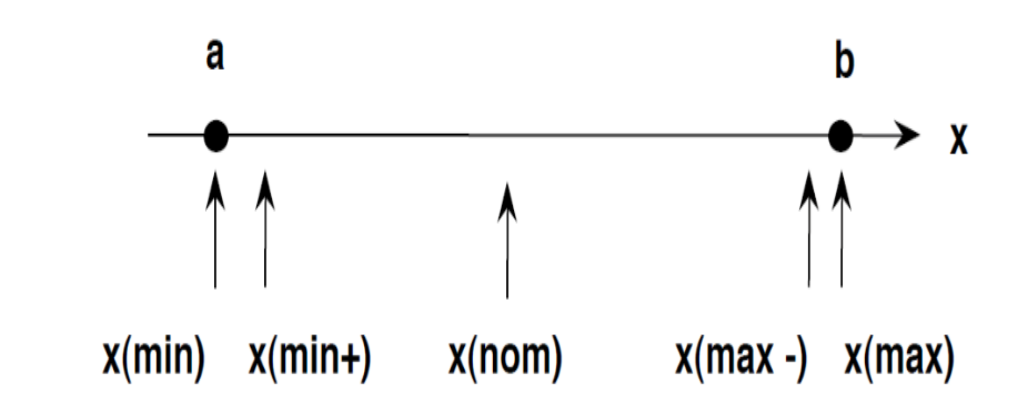
2.Then we made Test Cases

BOUNDARY VALUE ANALYSIS

- Boundary value analysis is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition).
- Used to find the errors at boundaries of input domain rather than finding those errors in the center of input.
- Uses same principal
  - Inputs & Outputs grouped into Classes
- For example; an Address text box which allows maximum 500 characters. So, writing test cases for each character once will be very difficult so that will choose boundary value analysis.

BOUNDARY VALUE ANALYSIS

- **Input: range of values**
  - Test Cases (valid) for the ends of the range
  - Test Cases (invalid) for conditions just beyond the ends
- The basic idea in boundary value testing is to select input variable values at their:
  - Minimum
  - Just above the minimum
  - A nominal value
  - Just below the maximum
  - Maximum



BOUNDARY VALUE ANALYSIS EXAMPLE

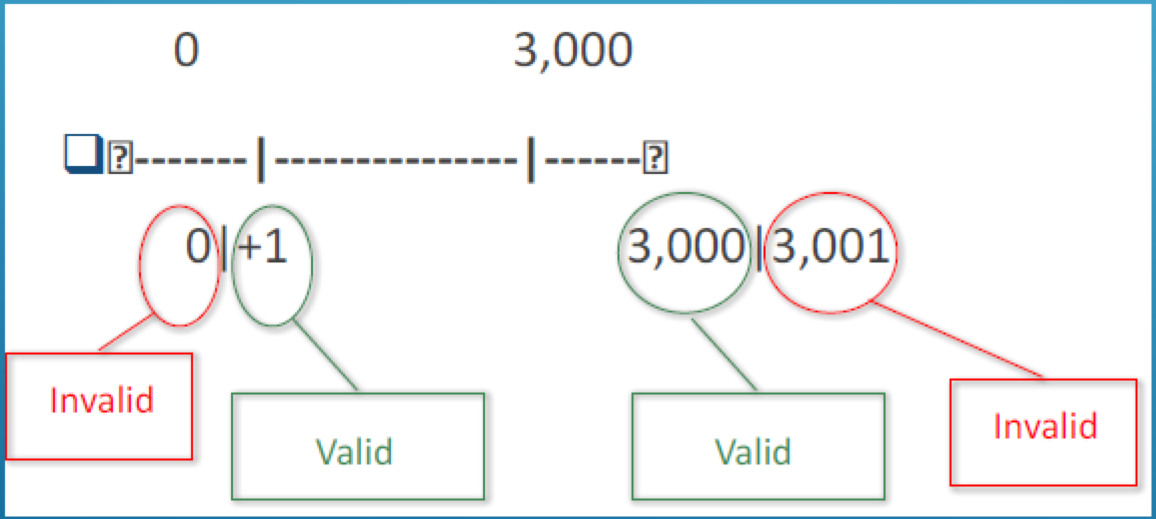
- ❑ If one application is accepting input range from 1 to 100, then test case design using Boundary value analysis will be as below:
  - One test case for exact boundary values of input domains each means 1 and 100.
  - One test case for just below boundary value of input domains each means 0 and 99.
  - One test case for just above boundary values of input domains each means 2 and 101.

BOUNDARY VALUE ANALYSIS EXAMPLE (1/3)

- Employees of an organization are allowed to get accommodation expenses while traveling on official tours. The program for validating expenses claims for accommodation has the following requirements
- There is an upper limit of Rs. 3,000 for accommodation expense claims
- Any claim above Rs. 3,000 should be rejected and cause an error message to be displayed
- All expense amount should be greater than zero and an error message to be displayed if this is not the case

BOUNDARY VALUE ANALYSIS EXAMPLE (2/3)

- Inputs: Accommodation Expense
- Boundaries of the Input Value
- Better to show Boundaries Graphically
  - Boundary: 0 < Expense ≤ 3,000



BOUNDARY VALUE ANALYSIS EXAMPLE (3/3)

Test Case ID	Input Value (Expense)	Boundary	Expected Output
1	0	0	Invalid claim, error message displayed
2	-1	0	Invalid claim, error message displayed
3	1	0	Valid claim, accepted
4	2999	3000	Valid claim, accepted
5	3000	3000	Valid claim, accepted
6	3000.01	3000	Invalid claim, error message displayed
7	4000	3000	Invalid claim, error message displayed

EQUIVALENCE PARTITIONING AND BOUNDARY VALUE ANALYSIS

Equivalence Partitioning and Boundary value analysis are linked to each other and can be used together at all levels of testing.

ERROR GUESSING

- Error guessing is a test case design technique where a test engineer uses experience to

  (i) guess the *types* and probable *locations* of *defects* and

  (ii) *design tests* specifically to *reveal the defects*.

- Few common mistakes that developers usually forget to handle:
  - Division by zero.
  - Handling null values in text fields.
  - Accepting the Submit button without any value.
  - File upload without attachment.
  - File upload with less than or more than the limit size.

ERROR GUESSING

- **Few strategies he can adopt for an error guessing are:**
  - Portions of the code with a high cyclomatic complexity are likely to have defects.
  - The code that has been recently added or modified can potentially contain defects.
  - Portions of code with prior defect history are likely to be prone to errors.
  - Parts of a system where new, unproven technology has been used is likely to contain defects.
  - Portions of the code for which the functional specification has been loosely defined can be more defective.
  - Code blocks that have been produced by novice developers can be defective.
  - If several developers worked on a particular part of the code, there is a possibility of misunderstanding among different developers and, therefore, there is a good possibility of errors in these parts of the code.
  - High-risk code will be more thoroughly tested.

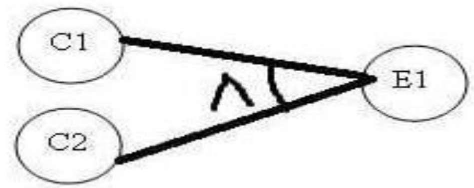
CAUSE EFFECT GRAPH

- Cause and effect graph is a dynamic test case writing technique. Here: **causes** are the input conditions and **effects** are the results of those input conditions.
- Cause-Effect Graph technique starts with a set of requirements and determines the minimum possible test cases for maximum test coverage which reduces test execution time and ultimately cost and helps to achieve desired application quality.

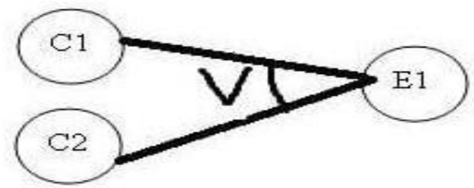
NOTATION FOR CAUSE EFFECT GRAPH

Identity Function: if C1 is 1, then E1 is 1. Else e is 0.

AND – For effect E1 to be true, both the causes C1 and C2 should be true

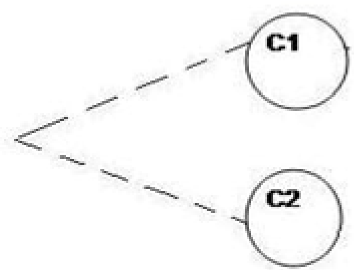


OR – For effect E1 to be true, either of causes C1 OR C2 should be true



NOT – For Effect E1 to be True, Cause C1 should be false

Exclusive constraint or **E-constraint:** This constraint exists between causes. It states that either c1 or c2 can be 1, i.e., both causes c1 and c2 cannot be 1 simultaneously.



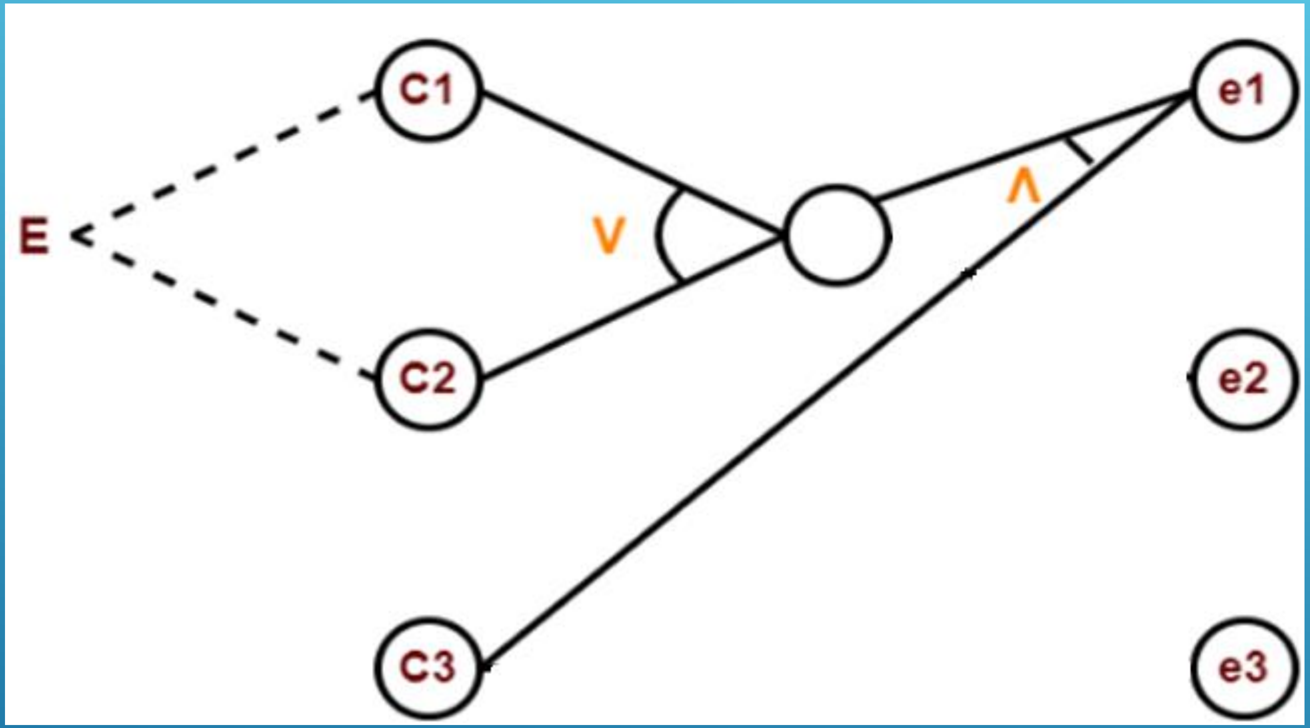
CAUSE EFFECT GRAPH EXAMPLE (1/5)

- **Situation:**
- The first character must be an “A” or a “B”.
- The second character must be a digit.
- If the first character is an “A” or “B” and the second character is a digit, print (“Hello World”)
- If the first character is incorrect (not an “A” or “B”), the message X must be printed.
- If the second character is incorrect (not a digit), the message Y must be printed.

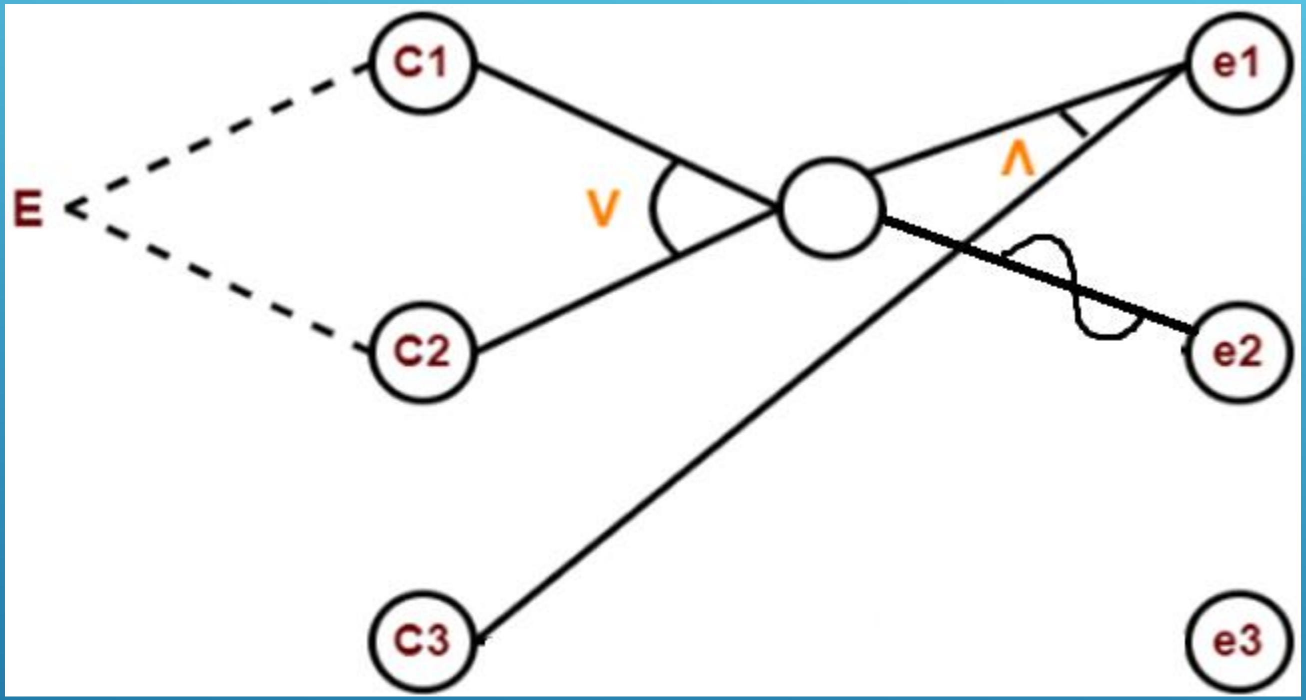
CAUSE EFFECT GRAPH EXAMPLE (2/5)

- **Solution:**
- **The causes for this situation are:**
  C1 – First character is A
  C2 – First character is B
  C3 –Second character is a digit
- **The effects (results) for this situation are**
  E1 – Print message “Hello World”
  E2 – Print message “X”
  E3 – Print message “Y”

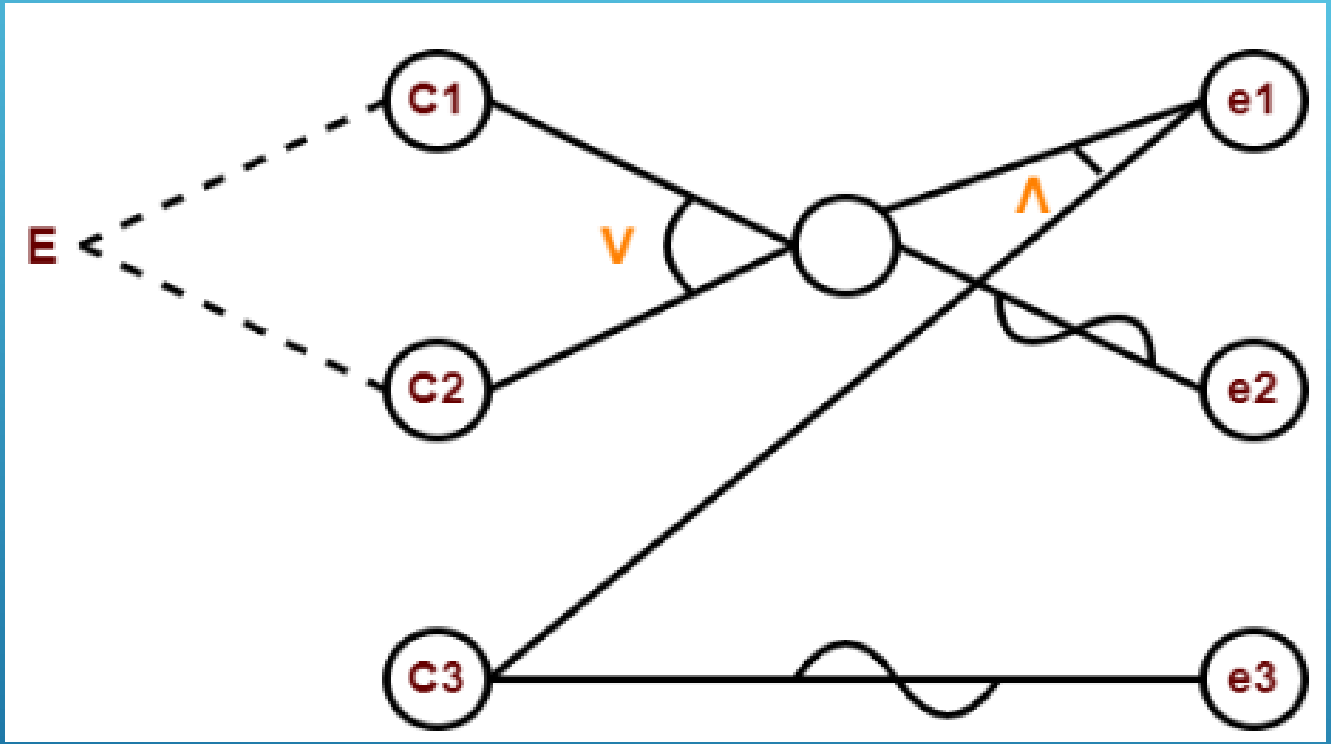
CAUSE EFFECT GRAPH EXAMPLE (3/5)



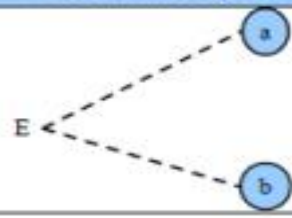
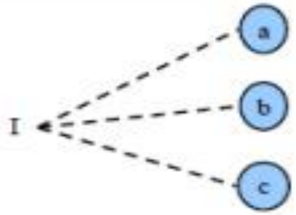
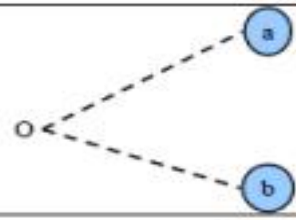
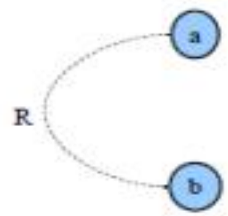
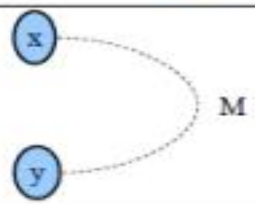
CAUSE EFFECT GRAPH EXAMPLE (4/5)



CAUSE EFFECT GRAPH EXAMPLE (5/5)



CONSTRAINTS SYMBOLS

Constraint Symbol	Definition
	The "E" (Exclusive) constraint states that both causes *a* and *b* cannot be true simultaneously.
	The "I" (Inclusive (at least one)) constraint states that at least one of the causes *a*, *b* and *c* must always be true *(a*, *b*, and *c* cannot be false simultaneously).
	The "O" (One and Only One) constraint states that one and only one of the causes *a* and *b* can be true.
	The "R" (Requires) constraint states that for cause *a* to be true, than cause *b* must be true. In other words, it is impossible for cause *a* to be true and cause *b* to be false.
	The "M" (mask) constraint states that if effect *x* is true; effect *y* is forced to false. (Note that the mask constraint relates to the effects and not the causes like the other constraints.

REFERENCES

Book:

The Art of Software Testing, Second Edition, Glenford J. Myers

Chapter 4