What is test case design?

TEST CASE is a set of conditions or variables under which a tester will determine whether a system under **test** satisfies requirements or works correctly. The process of developing **test cases** can also help find problems in the requirements or **design** of an application.

- A good test case design technique is crucial to improving the quality of the software testing process. This helps to improve the overall quality and effectiveness of the released software.

  Test case is a set of step-by-step instructions to verify something behaves as it is expected.

  Ex: Login page


amazon
Sign in
Email (phone for mobile accounts)
Password
Forgot your password?
Sign in
Keep me signed in. Details
New to Amazon?
Create your Amazon account

Introduction to Testing Design Strategies

- Testing Maturity Model serve as a learning tool, or framework, to learn about testing
- It introduces both the technical and managerial aspects of testing in a manner that allows for a natural evolution of the testing process, both on the personal and organizational levels
- Development of testing skills can be achieved at TMM levels 2-3
- TMM level 2 has three maturity goals, two of which are managerial in nature
- The technically oriented maturity goal at level 2 which calls for an organization to "institutionalize basic testing techniques and methods" addresses important and basic technical issues related to execution-based testing.

The Smart Tester

- Developers cannot prevent/eliminate all defects during development.
- It is the responsibility of the testers to design tests that (i) reveal defects, and (ii) can be used to evaluate software performance, usability, and reliability.
- Hence, testers must select a finite number of test cases, often from a very large execution domain.
- Unfortunately, testing is usually performed under budget and time constraints.
- Testers often are subject to enormous pressures from management and marketing because testing is not well planned, and expectations are unrealistic.
- The smart tester must plan for testing, select the test cases, and monitor the process to insure that the resources and time allocated for the job are utilized effectively.
- Proper education and training is needed to carry out these difficult tasks.
- Novice tester – use all possible inputs and exercise all possible software structures.
- However an informed and educated tester knows that is not a realistic or economically feasible goal.

Test Case Design Strategies

- Smart tester needs to maximize use of time and resources and for that he would develop ***effective test cases*** for execution-based testing.
- The ability to develop effective test cases is important to an organization evolving toward a higher-quality testing process.
- For example, if test cases are effective, there is,
  - a greater probability of detecting defects,
  - a more efficient use of organizational resources,
  - a higher probability for test reuse,
  - closer adherence to testing and project schedules and budgets, and
  - the possibility for delivery of a higher-quality software product.

What are the approaches a tester should use to design effective test cases?

- Two basis strategies are there :
  - Black box test strategies
  - White box test strategies

- Another approach might be for the tester to select test inputs at random, hoping that these tests will reveal critical defects.
- Randomly generated test i/p s have a poor performance record.
- The goal of the Smart Tester is to understand the functionality, input/output domain, and the environment of use for the code being tested.
- The tester must also knows how the code is being constructed?
- Finally, a smart tester needs to use knowledge of the types of defects that are commonly injected during development or maintenance of this type of software.
- Using this information, the smart tester must then intelligently select a subset of test Inputs that have the greatest possibility of revealing defects within the conditions and constraints placed on the testing process.

What is White Box Testing?

- **WHITE BOX TESTING** (also known as Clear **Box Testing**, Open **Box Testing**, Glass **Box Testing**, Transparent **Box Testing**, Code-Based **Testing** or Structural **Testing**) is a **software testing** method in which the internal structure/design/implementation of the item being tested is known to the **tester**.

- **WHITE BOX TESTING** is testing of a software solution's internal structure, design, and coding.
- In this type of testing, the code is visible to the tester.
- It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security.
- It is usually performed by developers.

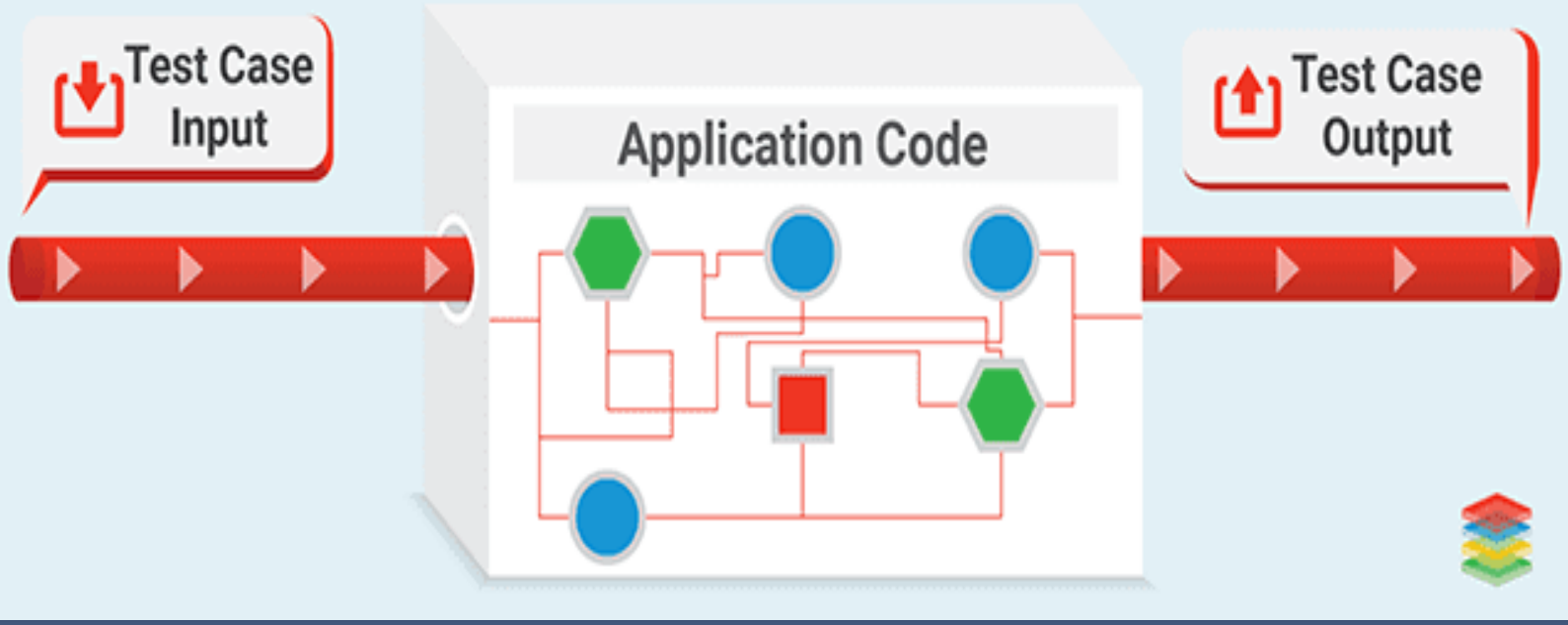
- In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.
- The tester chooses inputs to exercise paths through the code and determine the expected outputs.
- White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today.

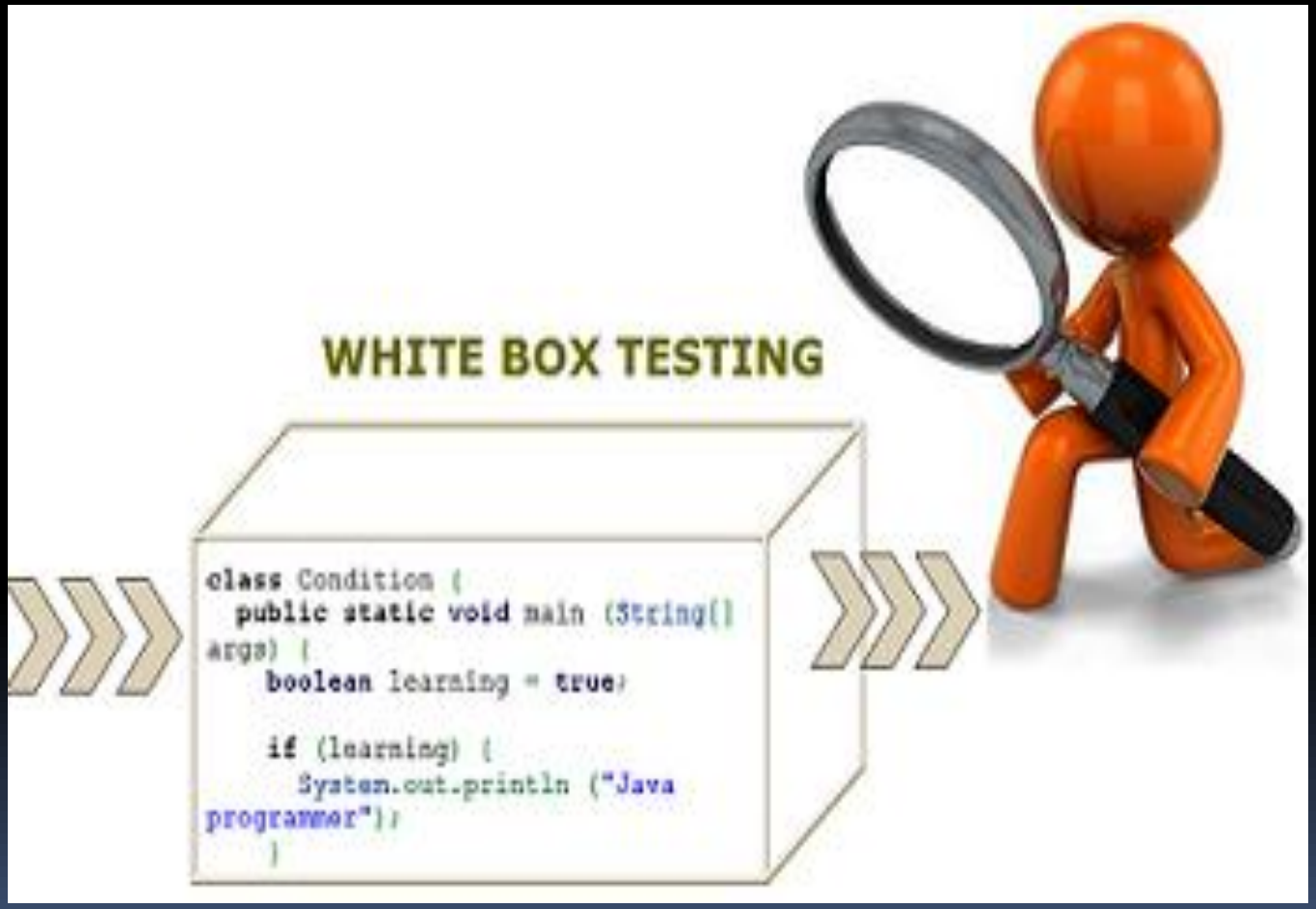
White Box Testing

Unit Testing

Integration Testing

System Testing



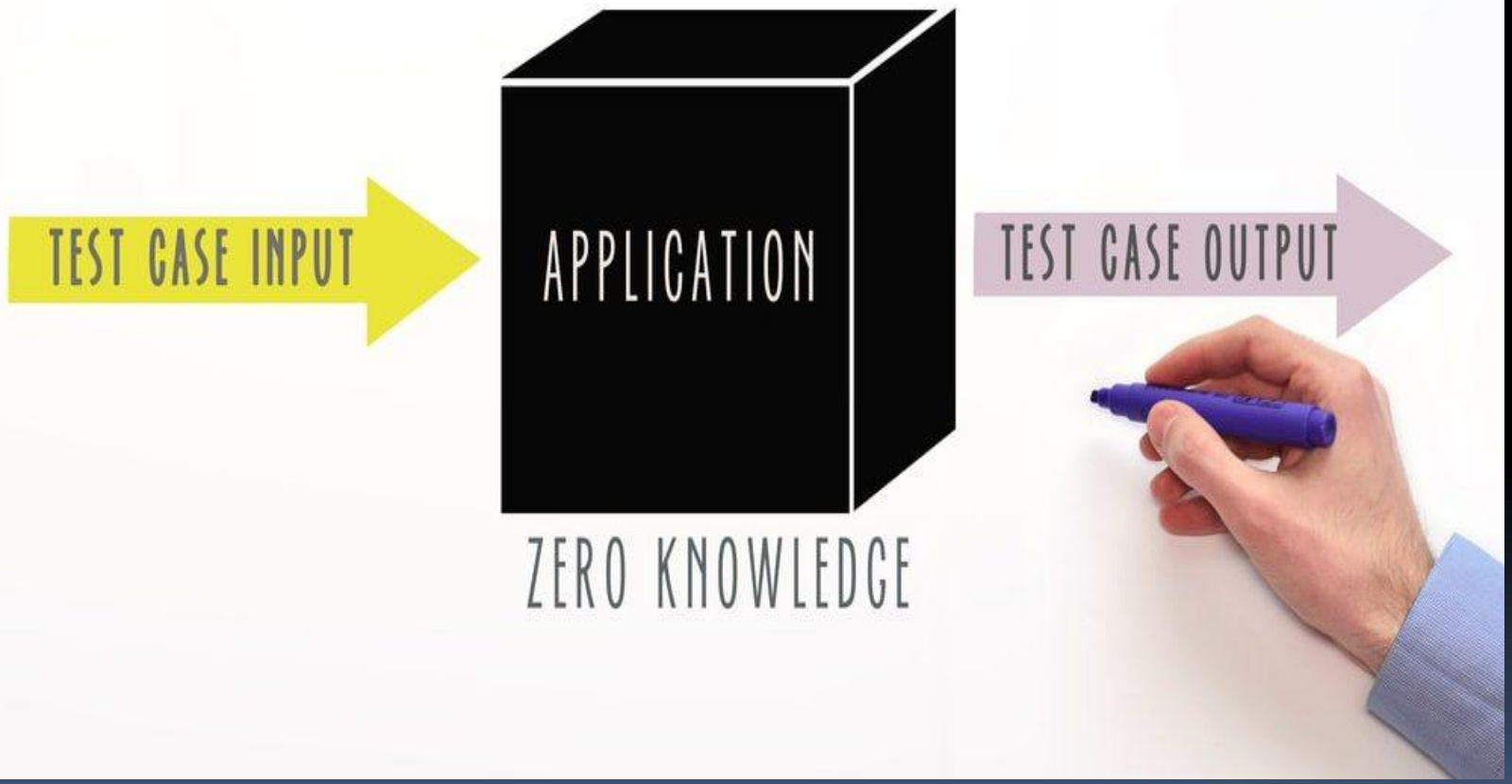

WHITE BOX TESTING
class Condition {
public static void main (String[]
args) {
boolean learning = true;
if (learning) {
System.out.println ("Java
programmer");
}

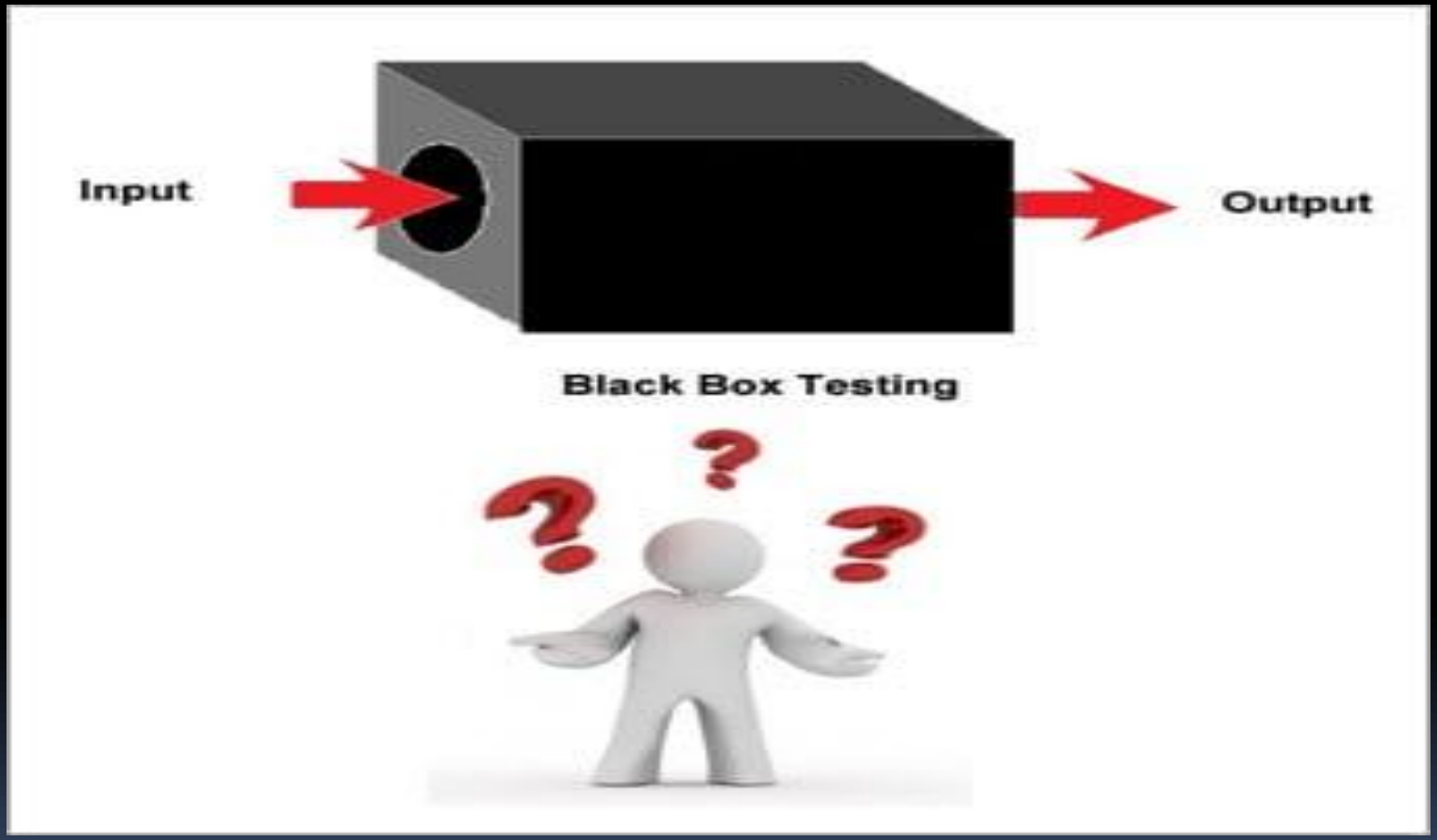
- Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.
- White-box testing's basic procedures require the tester to have an in-depth knowledge of the source code being tested.
- The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing.
- Once the source code is understood then the source code can be analyzed for test cases to be created.
- The following are the three basic steps that white-box testing takes in order to create test cases

- Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code and security specifications. This is the preparation stage of white-box testing to lay out all of the basic information.
- Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
- Output involves preparing final report that encompasses all of the above preparations and results.

- **BLACK BOX TESTING**, **also known** as Behavioral **Testing, is a** software **testing** method in which the internal structure/design/implementation of the item being tested is not **known** to the **tester**.
- **BLACK BOX TESTING** is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software.
- This type of testing is based entirely on software requirements and specifications.
- In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

BLACK-BOX TESTING




Input
Output
Black Box Testing

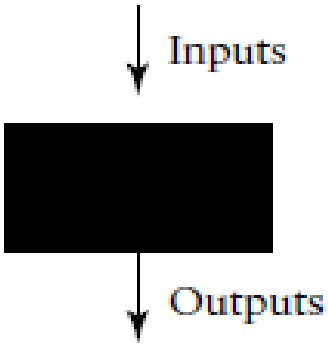
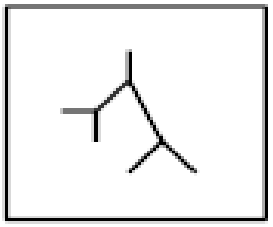
Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications Domain knowledge Defect analysis data	Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing
White box		High-level design Detailed design Control flow graphs Cyclomatic complexity	Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing

FIG. 4.1

The two basic testing strategies.

- Black box testing strategies
  - Software-under-test considered to be an opaque box
  - no knowledge of its inner structure
  - Size of the software-under-test can vary from a simple module, member function, or object cluster to a subsystem or a complete software system
  - description of behavior or functionality for the software-under-test may come from a formal specification, an Input/Process/Output Diagram (IPO), or a well-defined set of pre and post conditions
  - Hence, it is often called functional, or specification-based testing
  - This approach is especially useful for revealing requirements and specification defects
- White box testing strategies
  - Focuses on the inner structure of the software to be tested
  - To design test cases , the tester must have a knowledge of that structure
  - Code or pseudo code representation is available

- The tester selects test cases to exercise specific internal structural elements to determine if they are working properly.
- Time consuming, hence this strategy is usually applied to smaller-sized pieces of software such as a module or member function
- Especially useful for revealing design and code-based control, logic and sequence defects, initialization defects, and data flow defects.

- Smart tester knows that to achieve the goal of providing users with low-defect, high-quality software, ***both of these strategies should be*** used to design test cases.
- With a suite of test cases designed using both strategies the tester increases the chances of revealing the many different type of defects in the software-under-test.

Using the Black Box Approach to Test Case Design

Techniques of Black Box Testing

The following are the techniques employed while using Black box testing for a software application.

1. BVA or Boundary Value Analysis:

Boundary value analysis is another black box test design techniques and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.

These are widely used to pick the test cases in a systematic manner.

By using these techniques we could save lots of testing time and get the good test coverage.

Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions.

The Behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

- Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.
- A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.
- Tests can be designed to cover both valid and invalid boundary values. When designing test cases, a test for each boundary value is chosen.
- For each boundary, we test +/-1 in the least significant digit of either side of the boundary.
- Boundary value analysis can be applied at all test levels.

Example 1:

AGE Enter Age *Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

Minimum boundary value is 18
Maximum boundary value is 56
Valid Inputs: 18,19,55,56
Invalid Inputs: 17 and 57
Test case 1: Enter the value 17 (18-1) = Invalid
Test case 2: Enter the value 18 = Valid
Test case 3: Enter the value 19 (18+1) = Valid
Test case 4: Enter the value 55 (56-1) = Valid
Test case 5: Enter the value 56 = Valid
Test case 6: Enter the value 57 (56+1) =Invalid

- **Equivalence Partitioning (EP)**

  In Equivalence Partitioning, the test input data is partitioned into a number of classes having an equivalent number of data.

- The test cases are then designed for each class or partition. This helps to reduce the number of test cases.
- Dividing the test input data into a range of values and selecting one input value from each range is called **Equivalence Partitioning**. This is a black box test design technique used to calculate the effectiveness of test cases and which can be applied to all levels of testing from unit, integration, system testing and so forth.

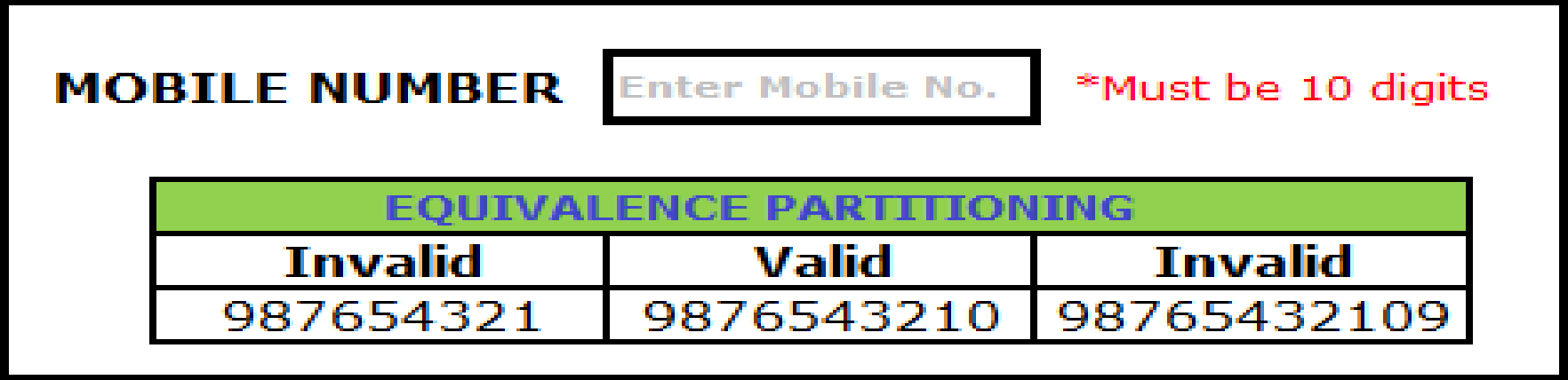
- We cannot test all the possible input domain values, because if we attempted this, the number of test cases would be too large. In this method, input data is divided into different classes, each class representing the input criteria from the equivalence class.
- We then select one input from each class.
- 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.

  Let's take a very basic and simple example to understand the *Equivalence Partitioning* concept:
- 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 one value below 1.
- One is for invalid data above upper limit i.e. any value above 100.

- Equivalence Partitioning is also known as Equivalence Class Partitioning.
- In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way.
- Hence selecting one input from each group to design the test cases.
- Each and every condition of particular partition (group) works as same as other.
- If a condition in a partition is valid, other conditions are valid too.
- If a condition in a partition is invalid, other conditions are invalid too.

- It helps to reduce the total number of test cases from infinite to finite.
- The selected test cases from these groups ensure coverage of all possible scenarios.
- Equivalence partitioning is applicable at all levels of testing.
- **Example on Equivalence Partitioning Test Case Design Technique:**
- **Example 1:**
- Assume, we have to test a filed which accepts a Mobile Number of ten digits.



1.Valid input: 10 digits
2.Invalid Input: 9 digits, 11 digits
3.Valid Class: Enter 10 digit mobile number = 9876543210
4.Invalid Class Enter mobile number which has less than 10 digits = 987654321
5.Invalid Class Enter mobile number which has more than 11 digits = 98765432109

State-Based testing

What is State Transition in Testing?

- State Transition testing is defined as the software testing technique in which changes in input conditions cause's state changes in the Application under Test (AUT).
- It is a black box testing technique in which the tester analyzes the behavior of an application under test for different input conditions in a sequence. In this technique, tester provides both positive and negative input test values and record the system behavior.

- It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- **State Transition Testing Technique** is helpful where you need to **test different system transitions**

When to Use State Transition?

- This can be used when a tester is testing the application for a finite set of input values.
- When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behavior for a sequence of input values.
- When the system under test has a dependency on the events/values in the past.

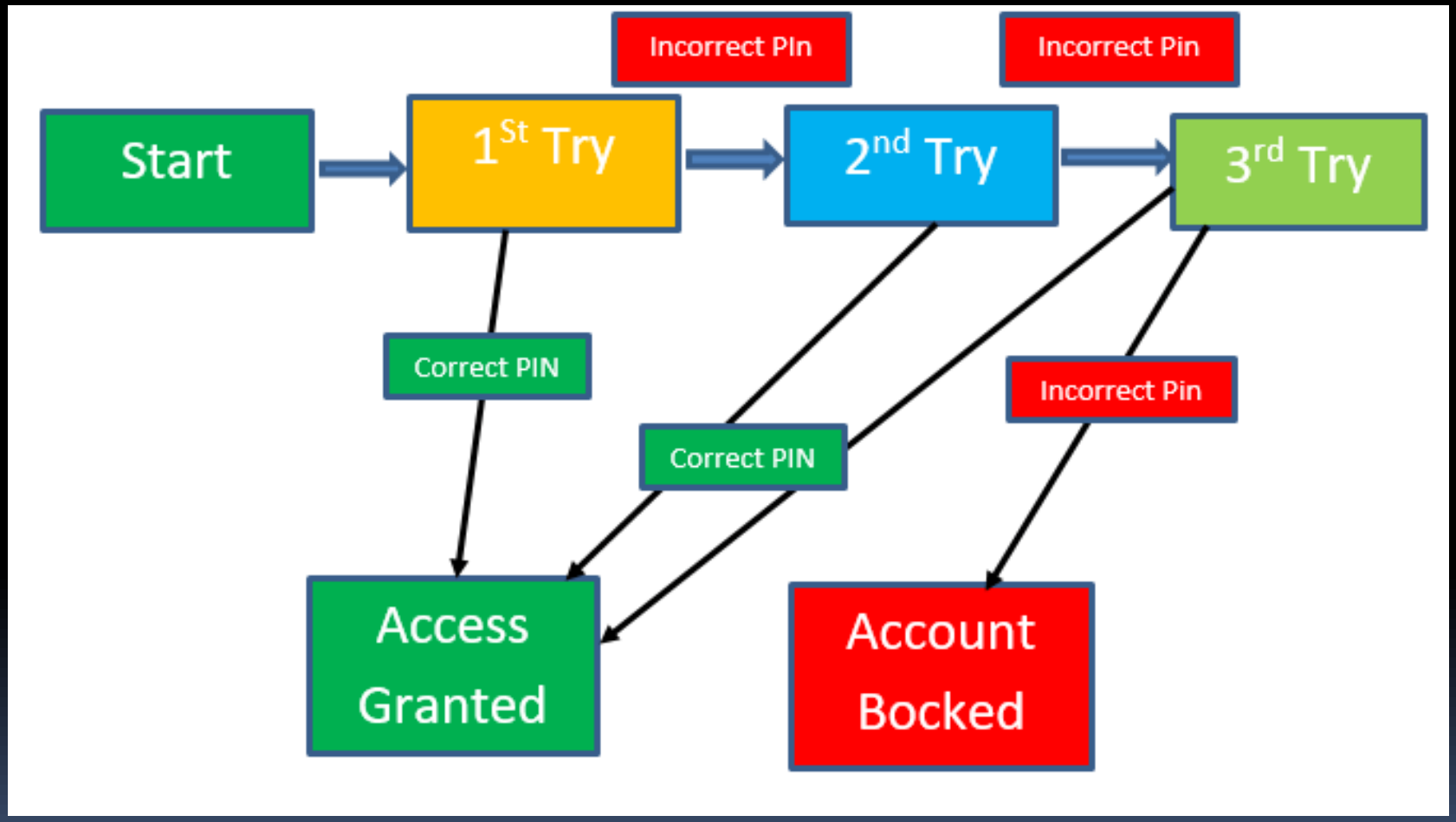
State Transition Diagram and State Transition Table

- There are two main ways to represent or design state transition, State transition diagram, and state transition table.
- In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.
- In state transition table all the states are listed on the left side, and the events are described on the top. Each cell in the table represents the state of the system after the event has occurred. It is also called State Table. It is useful in identifying invalid transitions.

- **How to Make a State Transition (Examples of a State Transition)**

Example 1:

- Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.
- In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked

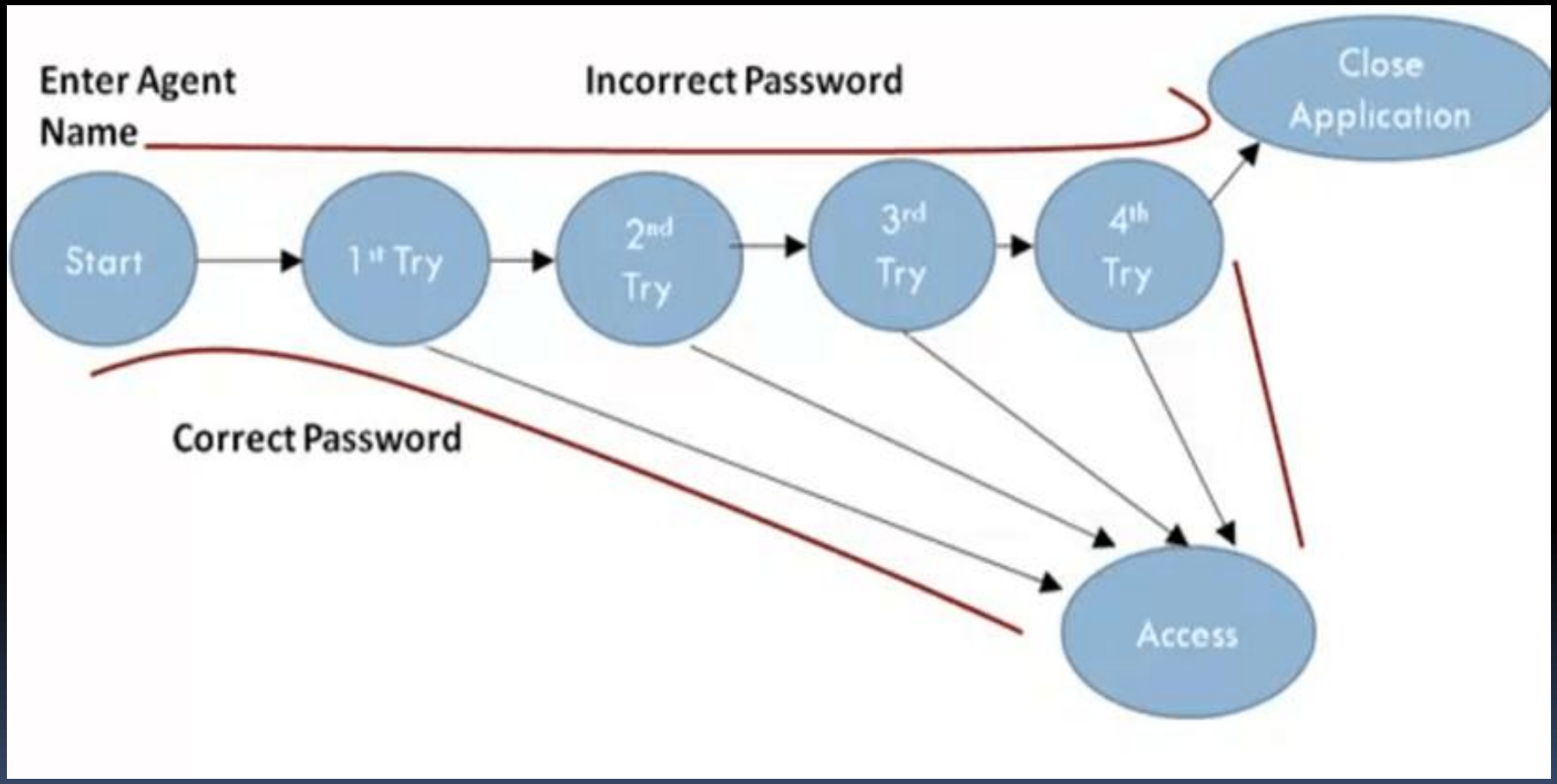

Incorrect PIn
Incorrect Pin
Start
1St Try
2nd Try
3rd Try
Correct PIN
Incorrect Pin
Correct PIN
Access Granted
Account Bocked

- In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

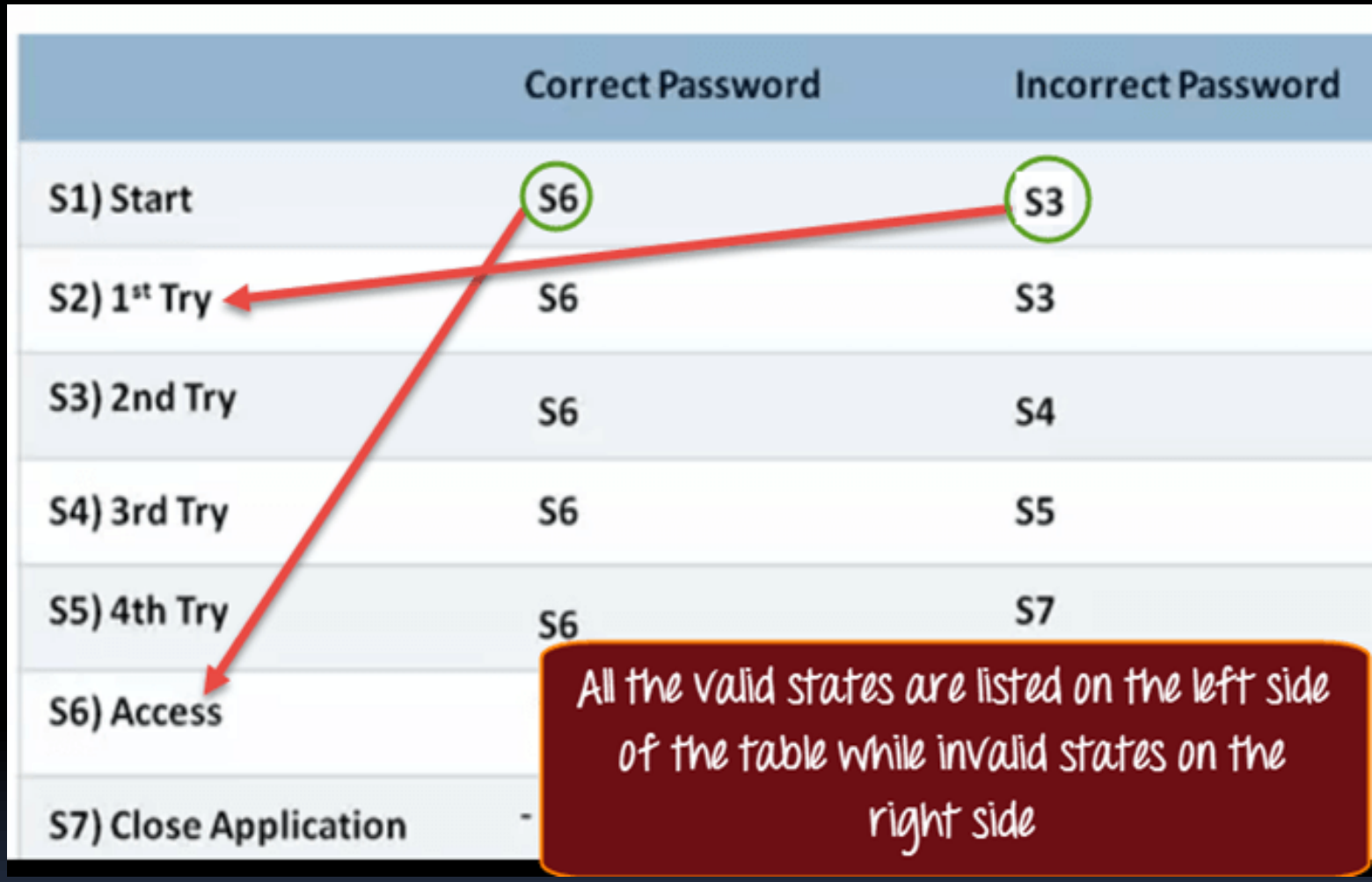
Four Parts Of State Transition Diagram

There are 4 main components of the State Transition Model as below

1) **States** that the software might get
2) **Transition** from one state to another
3) **Events** that origin a transition like closing a file or withdrawing money.
4) **Actions** that result from a transition (an error message or being given the cash.)


Enter Agent Name
Incorrect Password
Close Application
Start
1st Try
2nd Try
3rd Try
4th Try
Correct Password
Access

- It gives you the access to the application with correct password and login name, but what if you entered the wrong password.
- The application allows three attempts, and if users enter the wrong password at 4th attempt, the system closes the application automatically.
- The State Graphs helps you determine valid transitions to be tested. In this case, testing with the correct password and with an incorrect password is compulsory. For the test scenarios, log-in on 2nd, 3rd and 4th attempt anyone could be tested.



You can use State Table to determine invalid system transitions.

- In a State Table, all the valid states are listed on the left side of the table, and the events that cause them on the top.
- Each cell represents the state system will move to when the corresponding event occurs.
- For example, while in S1 state you enter a correct password you are taken to state S6 (Access Granted). Suppose if you have entered the wrong password at first attempt you will be taken to state S3 or 2nd Try.

- Likewise, you can determine all other states.
- Two invalid states are highlighted using this method. Suppose you are in state S6 that is you are already logged into the application, and you open another instance of flight reservation and enter valid or invalid passwords for the same agent. System response for such a scenario needs to be tested.

Summary:

- State Transition testing is defined as the testing technique in which changes in input conditions cause's state changes in the Application under Test.
- In Software Engineering, State Transition Testing Technique is helpful where you need to test different system transitions.
- Two main ways to represent or design state transition, State transition diagram, and State transition table.
- n state transition diagram the states are shown in boxed texts, and the transition is represented by arrows.

- In state transition table all the states are listed on the left side, and the events are described on the top.
- This main advantage of this testing technique is that it will provide a pictorial or tabular representation of system behavior which will make the tester to cover and understand the system behavior efficiently.
- T he main **Disadvantage** of this testing technique is that we can't rely in this technique every time. For example, if the system is not a finite system (not in sequential order), this technique cannot be used.

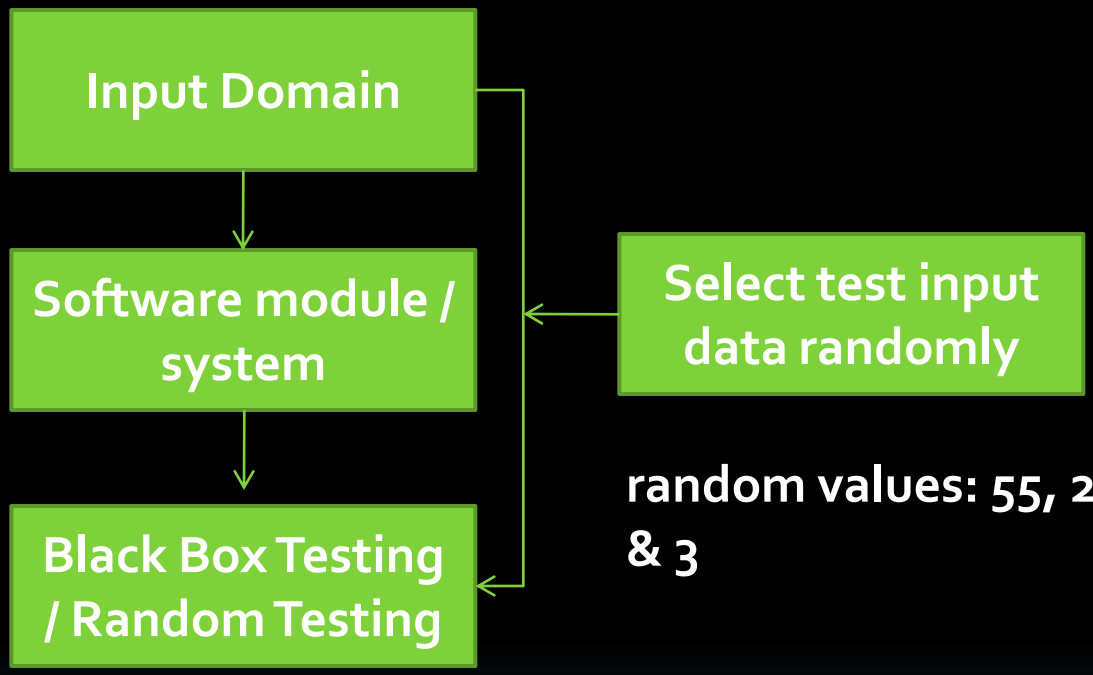
- **Advantages: 1.**This testing technique will provide a pictorial or tabular representation of system behavior which will make the tester to cover and understand the system behavior effectively.
- 2.By using this testing, technique tester can verify that all the conditions are covered, and the results are captured

Random testing:

- It is a black-box software **testing** technique where programs are tested by generating **random**, independent inputs. Results of the output are compared against software specifications to verify that the **test** output is pass or fail.
- Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests.
- Random testing is the testing technique done with the independence input. it is generated to check the output of the application is correct or not .
- There strength is defects are find very quickly and easily, it only find basic bugs.
  weakness is bugs are not identified in regular intervals and most of the tester like white box testing compare to this testing technique

Random testing

Example: **all positive integers between 1 & 100**



Given this approach, some of the issues that remain open are the following:

Are the three values adequate to show that the module meets its specification when the tests are run? Should additional or fewer values be used to make the most effective use of resources?

- Are there any input values, other than those selected, more likely to reveal defects? For example, should positive integers at the beginning or end of the domain be specifically selected as inputs?
- Should any values outside the valid domain be used as test inputs? For example, should test data include floating point values, negative values, or integer values greater than 100?
- Use of random test inputs may save some time and effort
- But, selecting test inputs randomly has very little chance of producing an effective set of test data

Random Testing Steps:

- Random Inputs are identified to be evaluated against the system.
- Test Inputs are selected independently from test domain.
- Tests are Executed using those random inputs.
- Record the results and compare against the expected outcomes.
- Reproduce/Replicate the issue and raise defects, fix and retest.

Random testing is a testing technique where programs are tested by generating random and independent inputs. The results of output generated are compared with the software specifications to verify if the result is correct or not. There are some strengths and weakness of random testing.

The strength of random testing are:

- It is inexpensive to use
- It does not have any bias
- The bugs are found very easily and quickly
- If software is used properly it will find the bugs.

The weakness of this testing is:

- It is capable of finding only basic bugs
- It is precise when specifications are imprecise.
- This technique compares poorly with other techniques to find the bugs
- This technique will create a problem for continuous integration if different inputs are randomly selected on each test.
- Some think that white box testing is better than this random testing technique

Random Testing Characteristics

- It is performed where defects in a software application is not identified by the regular intervals.
- Random input is used to test the system performance and its reliability.
- Saves time and effort than actual tests.
- Other testing methods are not used.

The common example of Random testing is: use of random integers to test the software function that returns the results based on those integers. Specifically when dealing with integers or other types of variables. Random testing is random as a set of random inputs that are used, in other words testers are bound to choose set of integers rather than infinite set.

Requirements Based Testing

What is Requirements Based Testing?

- The process of requirements based testing deals with validating whether the requirements are complete, consistent , unambiguous, complete and logically connected. With such requirements, we can proceed to develop test cases to ensure that the test cases full fill all the requirements.
- Testing in this technique revolves around requirements. The strategy of Requirement based testing is to integrate testing throughout the life cycle of the software development process, to assure quality of the Requirement Specification. The aim is defect prevention than defect detection.

Taking Requirement Based testing into account,

testing is divided into the following types of activity :

Stages in Requirements based Testing:

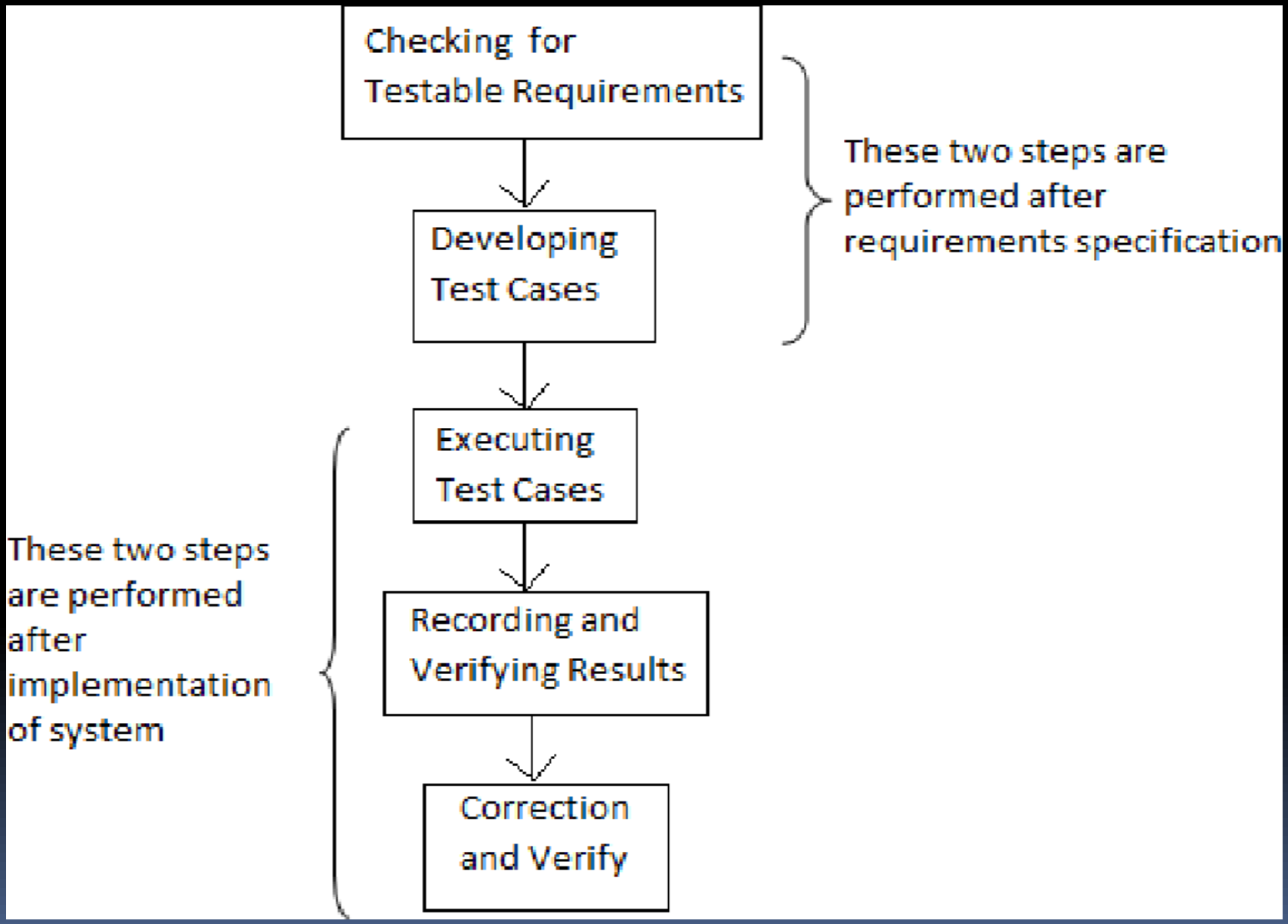
- **Define Test Completion Criteria :** Testing should be defined in quantifiable terms. The goal is considered to be achieved only when test coverage is 100%.
- **Design Test Cases :** Test cases must be in accordance with requirements specification.
- **Build Test Cases :**Join the logical parts together to form/build test cases .
- **Execute Test Cases :**Execute the test cases to evaluate the results.
- **Verify Test Results :**Check whether actual results deviate from the expected ones.
- **Manage Test Library :**Test manager is responsible for monitoring the test case executions, that is, the tests passed or failed, or to ascertain whether all tests have been successfully performed.
- **Track and Manage Defects -** Any defects detected during the testing process goes through the defect life cycle and are tracked to resolution. Defect Statistics are maintained which will give us the overall status of the project

Why Requirements are Critical :

- Various studies have shown that software projects fail due to the following reasons:
- Incomplete requirements and specifications.
- Frequent changes in requirements and specifications
- When there is lack of user input to requirements
- So the requirements based testing process addresses each of the above issues as follows :
- The Requirements based testing process starts at the very early phase of the software development, as correcting issues/errors is easier at this phase.
- It begins at the requirements phase as the chances of occurrence of bugs have its roots here.
- It aims at quality improvement of requirements. Insufficient requirements leads to failed projects.

Requirements Testing process:

- Testing must be carried out in a timely manner.
- Testing process should add value to the software life cycle, hence it needs to be effective.
- Testing the system exhaustively is impossible hence the testing process needs to be efficient as well.
- Testing must provide the overall status of the project, hence it should be manageable.


Checking for Testable Requirements
Developing Test Cases
Executing Test Cases
Recording and Verifying Results
Correction and Verify
These two steps are performed after requirements specification
These two steps are performed after implementation of system

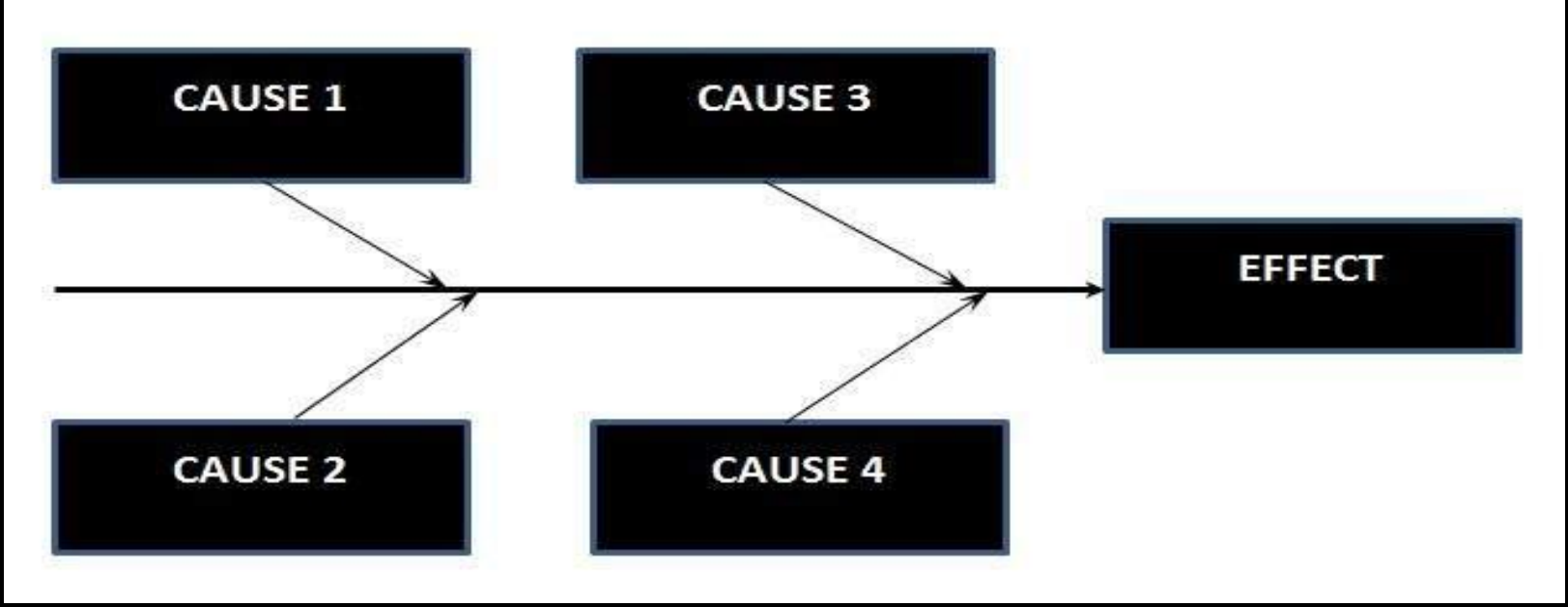
Cause-and-effect graphing

Cause Effect Graph Cause Effect Graph is a black box testing technique. … that aids in choosing test cases that logically relate **Causes** (inputs) to **Effects** (outputs) to produce test cases. A "**Cause**" stands for a separate input condition that fetches about an internal change in the system.. It is also known as Ishikawa diagram as it was invented by Kaoru Ishikawa or fish bone diagram because of the way it looks.

Cause-effect graph which underlines the relationship between a given result and all the factors affecting the result. It is used to write dynamic test cases.

- A “Cause” stands for a separate input condition that fetches about an internal change in the system. An “Effect” represents an output condition, a system transformation or a state resulting from a combination of causes.

- The dynamic test cases are used when code works dynamically based on user input. For example, while using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.
- Cause-Effect graph technique is based on a collection of requirements and used to determine minimum possible test cases which can cover a maximum test area of the software.
- The main advantage of cause-effect graph testing is, it reduces the time of test execution and cost.



The Cause-Effect Diagram can be used under these Circumstances:

1.To determine the current problem so that right decision can be taken very fast.

2.To narrate the connections of the system with the factors affecting a particular process or effect.

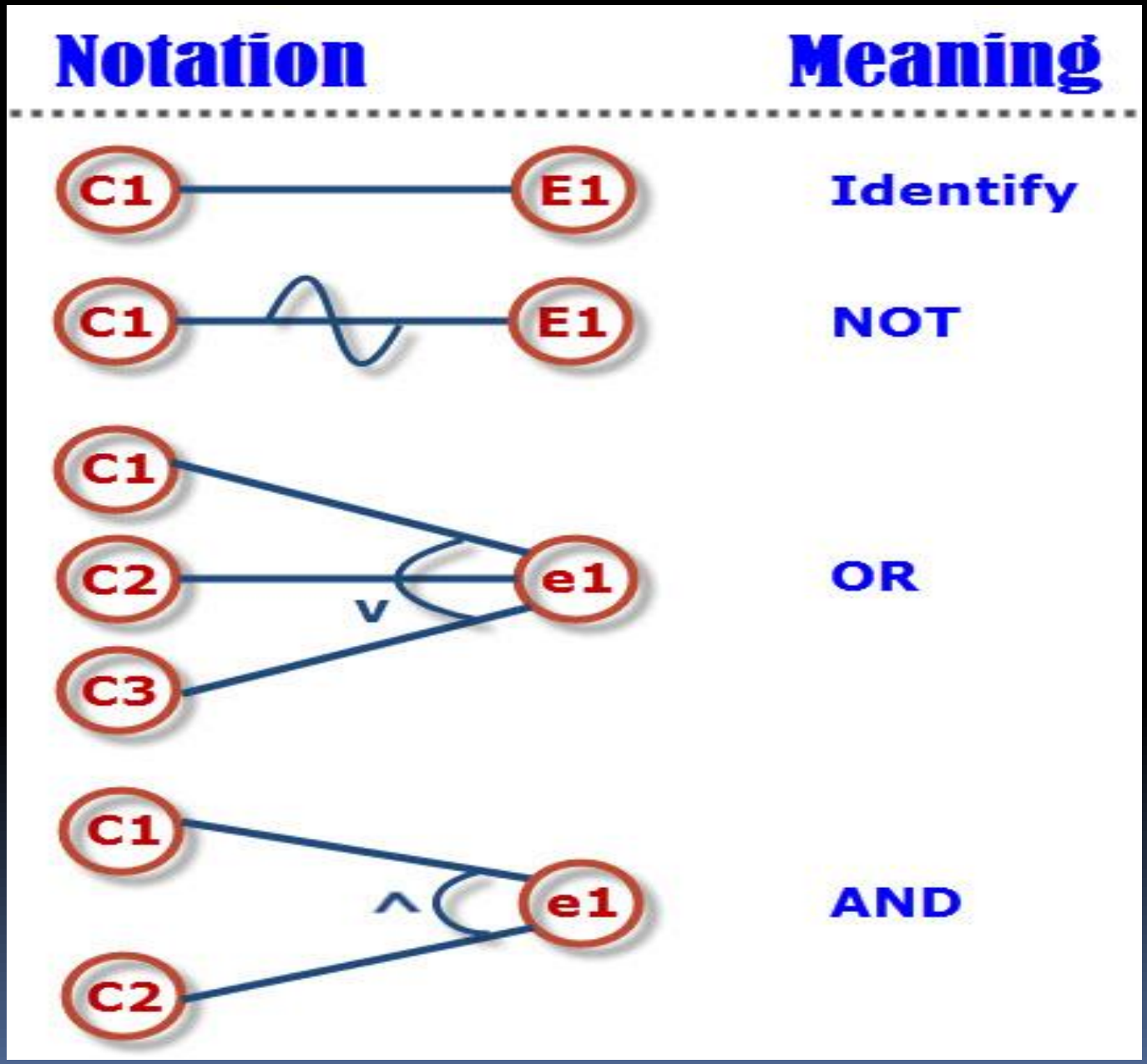
3.To recognize the probable root causes, the cause for a exact effect, problem, or outcome.

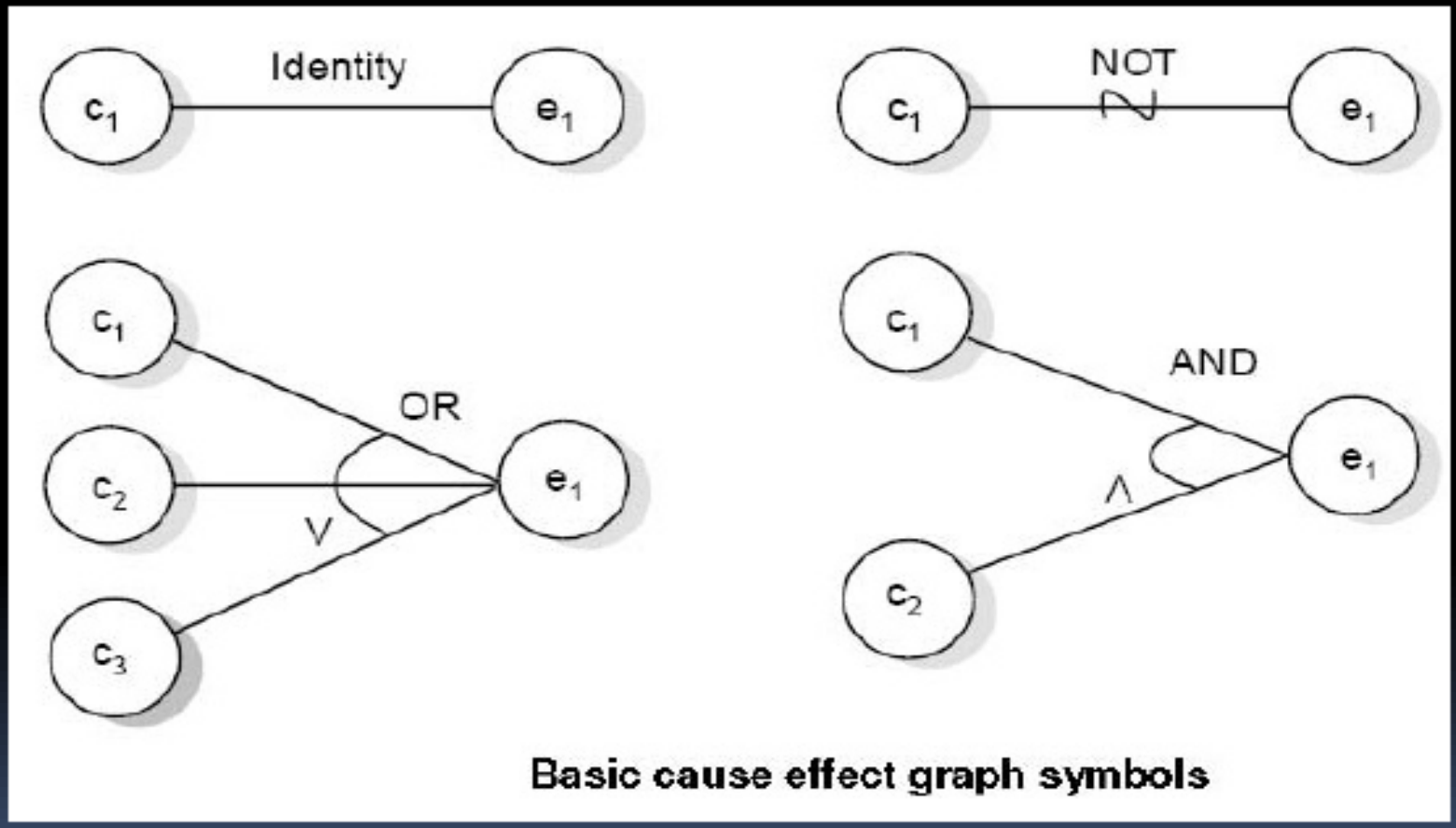
Benefits of making cause-Effect Diagram

- It finds out the areas where data is collected for additional study.
- It motivates team contribution and uses the team data of the process.
- Uses synchronize and easy to read format to diagram cause-and-effect relationships.
- Point out probable reasons of difference in a process.
- It enhances facts of the procedure by helping everyone to learn more about the factors at work and how they relate.
- It assists us to decide the root reasons of a problem or quality using a structured approach.

- The dynamic test cases are used when code works dynamically based on user input.
- For example, while using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message.
- In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.
- The main advantage of cause-effect graph testing is, it reduces the time of test execution and cost.
- Cause-Effect graph technique is based on a collection of requirements and used to determine minimum possible test cases which can cover a maximum test area of the software.

- This technique aims to reduce the number of test cases but still covers all necessary test cases with maximum coverage to achieve the desired application quality.
- Cause-Effect graph technique converts the requirements specification into a logical relationship between the input and output conditions by using logical operators like AND, OR and NOT.

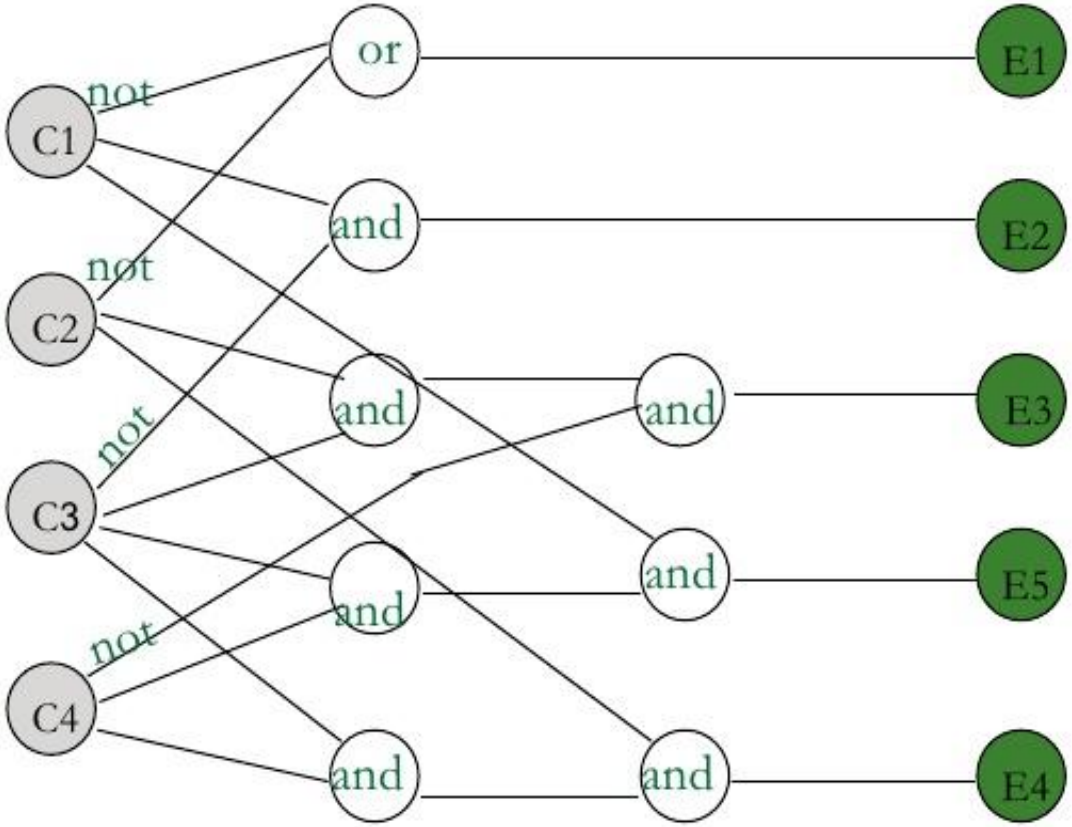

Notation
Meaning
C1
E1
Identify
C1
E1
NOT
C1
C2
C3
v
e1
OR
C1
C2
^
e1
AND



Basic cause effect graph symbols

- Just assume that each node having the value 0 or 1 where 0 shows the 'absent state' and 1 shows the 'present state'. The identity function states when c1 = 1, e1 = 1 or we can say if c0 = 0 and e0 = 0.
- The NOT function states that, if C1 = 1, e1= 0 and vice-versa. Likewise, OR function states that, if C1 or C2 or C3 = 1, e1 = 1 else e1 = 0. The AND function states that, if both C1 and C2 = 1, e1 = 1, else e1 = 0.
- The AND and OR functions are permitted to have any number of inputs.

Example: Cause-Effect Graph



Compatibility Testing

What is Compatibility?

- Compatibility is nothing but the capability of existing or living together. In normal life, Oil is not compatible with water, but milk can be easily combined with water.
- **What is Compatibility Testing?**
- Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.

Compatibility testing is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments. It can be of two types - forward compatibility testing and backward compatibility testing.

- Operating system Compatibility Testing - Linux , Mac OS, Windows
- Database Compatibility Testing - Oracle SQL Server
- Browser Compatibility Testing - IE , Chrome, Firefox
- Other System Software - Web server, networking/ messaging tool, etc.

What is Compatibility testing?

- Compatibility testing is a non-functional testing method primarily done to ensure customer satisfaction. This testing process will ensure that the software is compatible across operating systems, hardware platforms, web browsers, etc.
- The testing also works as validation for compatibility requirements that have been set at the planning stage of the software.
- The process helps in developing software that has the ability to work seamlessly across platforms and hardware without any trouble

- In today's competitive world, it is important that the software or the products released to the buyers reflect true value for the amount they incur to buy or use the product.
- Thorough testing of the products helps create quality products that provide value for money. Various software tests are performed at different stages of software development and testing is also conducted on the finished product, prior to its release.
- This testing is done to ensure a competitive edge in terms of quality, compatibility, cost, and delivery for the end product before it is delivered.

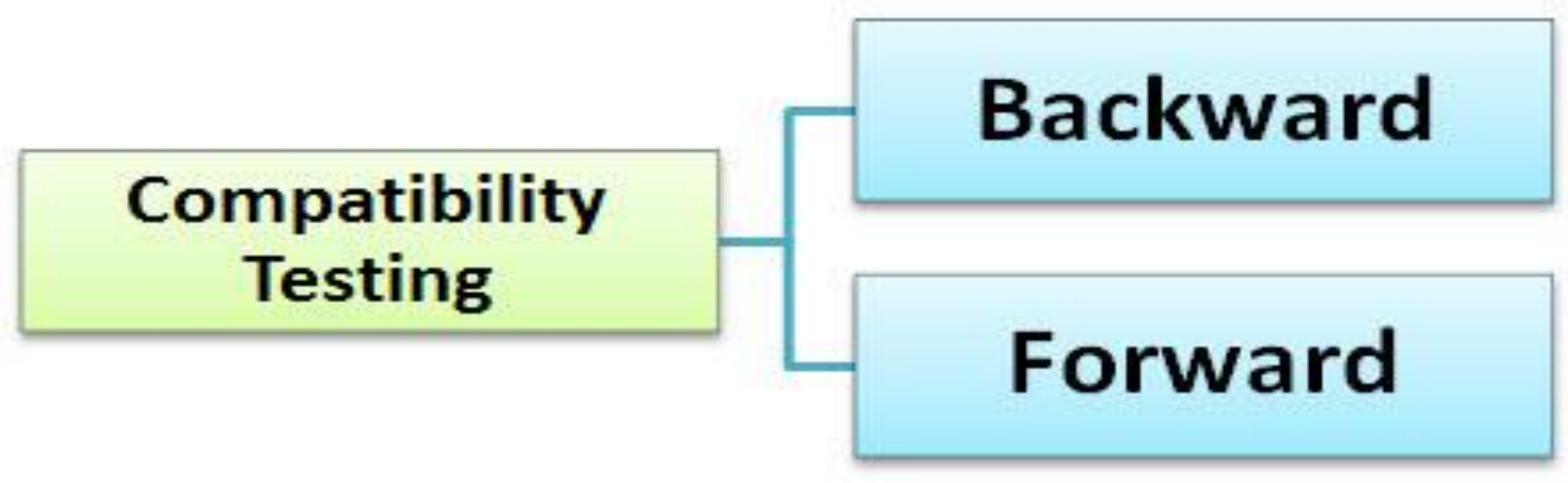
- Compatibility testing helps ensure complete customer satisfaction as it checks whether the application performs or operates as expected for all the intended users across multiple platforms.
- This non-functional testing is performed to ensure compatibility of a system, application, or website built with various other objects such as other web browsers, databases, hardware platforms, users, operating systems, mobile devices & networks etc.
- It is conducted on the application to evaluate the application's compatibility with different environments. It can be performed either through automation tools or it can be conducted manually.

Need for Compatibility Testing:

- Software applications released should be of high quality and compatible with all hardware, software, OS, platforms, etc.
- which is achieved through opting for compatibility testing. Compatibility can be ensured through adopting compatibility testing, which detects for any errors before the product is delivered to the end user.
- This testing establishes or confirms that the product meets all the requirements set and agreed upon by both the developer and the end user.
- This stable or quality product in turn improves the reputation of the firm and propels the company to success. It is also true that quality products improve sales and marketing efforts and bring delight to the customer.

- Moreover, an efficient compatibility test effort ensures real compatibility among different computing environments.
- In addition, a truly dynamic compatibility testing also confirms the workability and stability of the software that is of much importance before its release.

Types of Compatibility Testing

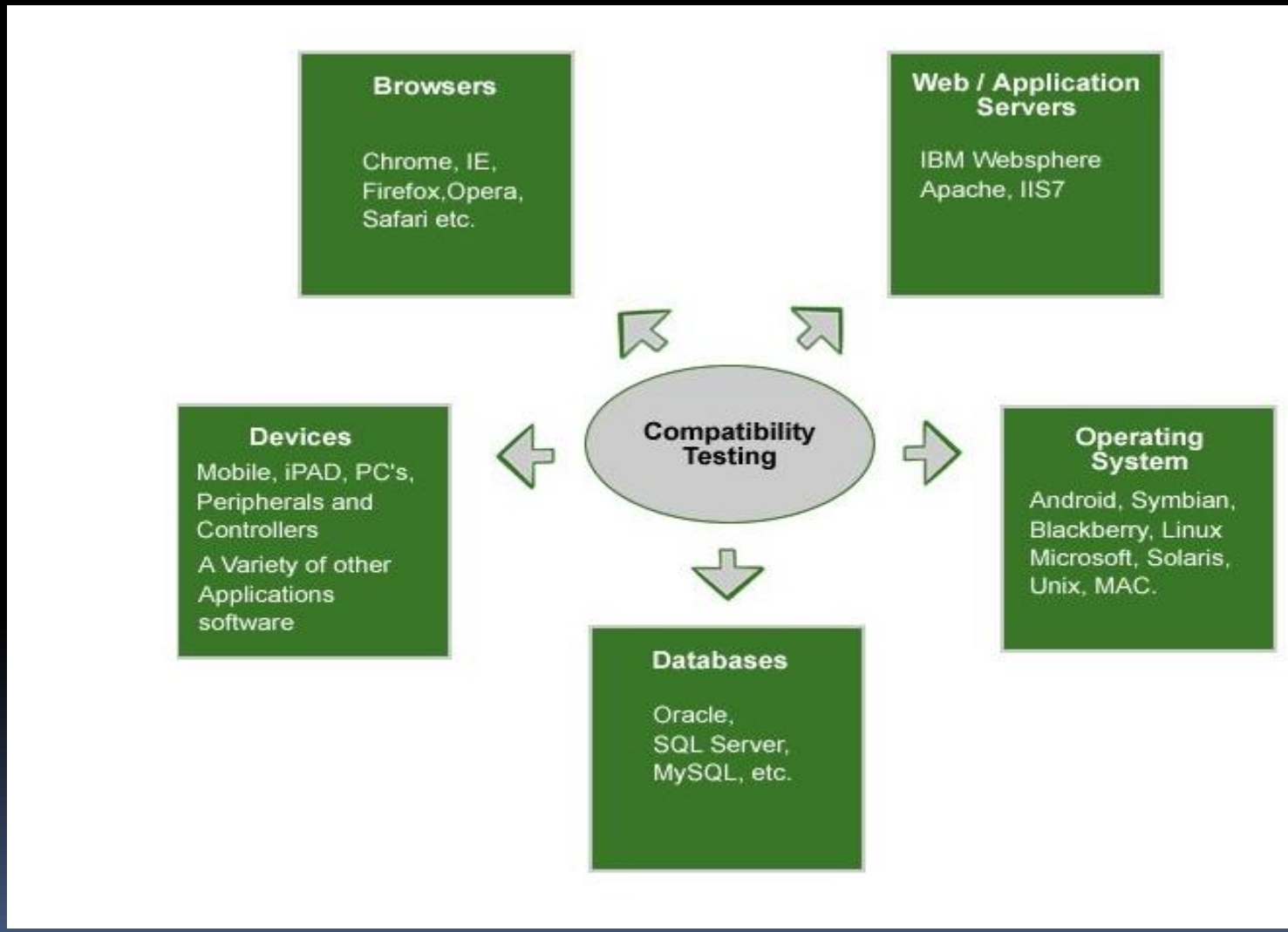


- ***#1)*** **Compatibility** ***Forward testing:*** This type of testing verifies that the software is compatible with the newer or upcoming versions, and is thus named as forward compatible.
- ***#2)*** **Compatibility** ***Backward testing*** checks whether the mobile app has been developed for the latest versions of an environment also work perfectly with the older version. The behavior of the new hardware/software has been matched against the behavior of the old hardware/software

- Compatibility type of testing can be performed on operating systems, databases, systems software, browsers, and mobile applications. The mobile app testing is performed across various platforms, devices, and networks.

Process of Compatibility Testing

- The compatibility test is conducted under different hardware and software application conditions, where the computing environment is important, as the software product created must work in a real-time environment without any errors or bugs.
- Some of the main computing environments are the operating systems, hardware peripherals, browsers, database content, computing capacity, and other related system software if any.


Browsers
Chrome, IE, Firefox,Opera, Safari etc.
Web / Application Servers
IBM Websphere Apache, IIS7
Devices
Mobile, iPAD, PC's, Peripherals and Controllers
A Variety of other Applications software
Compatibility Testing
Operating System
Android, Symbian, Blackberry, Linux Microsoft, Solaris, Unix, MAC.
Databases
Oracle, SQL Server, MySQL, etc.

user Documentation Testing

- Documentation for Software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing, etc. This section includes the description of some commonly used documented artifacts related to Software development and testing, such as:
- Test Plan
- Requirements
- Test Cases

USER DOCUMENTATION TESTING

- It is a type of **non-functional testing**.
- Any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results'. Documentation is as important to a product's success as the product itself. If the documentation is poor, non-existent, or wrong, it reflects on the quality of the product and the vendor.
- As per the IEEE Documentation describing , the testing of a system or component, include test case specification, test incident report, test plan, test procedure, test report. Hence the testing of all the above mentioned documents is known as documentation testing.

- This is one of the most cost effective approaches to testing. If the documentation is not right: there will be major and costly problems. The documentation can be tested in a number of different ways to many different degrees of complexity. These range from running the documents through a spelling and grammar checking device, to manually reviewing the documentation to remove any ambiguity or inconsistency.
- Documentation testing can start at the very beginning of the software process and hence save large amounts of money, since the earlier a **defect** is found the less it will cost to be fixed.
- The documentation testing generally includes two methods involving varying degree of complexity.

- Product documentation is a critical part of the final product.
- Poor documentation can affect the product or company reputation.[3]
- Documentation is about the testing of all the documents created prior and after the testing of software.[4] Any delay in the testing of the document will increase the cost. Some common artifacts about software development and testing can be specified as test cases, test plans, requirements, and traceability matrices.

- User Documentation covers all the manuals, user guides, installation guides, setup guides, read me files, software release notes, and online help that are provided along with the software to help the end user to understand the software system. User Documentation Testing should have two objectives:-
- **Method-1:** Checking of spellings and grammar in the documents using available tools.
- **Method-2:** Manual review of documents to discover errors, ambiguities or inconsistencies.

- This testing is plays a vital role as the users will refer this document when they start using the software at their location a badly written document can put off a user and bias them against the product even the product offers rich functionality.
- Defects found in the user documentation need to be tracked to closure like any regular software defect. Because these documents are the first interactions the users have with the product.
- A good User Documentation aids in reducing customer support calls. The effort and money spend on this effort would form a valuable investment in the long run for the organization.

Key Target Areas for testing of documentation:

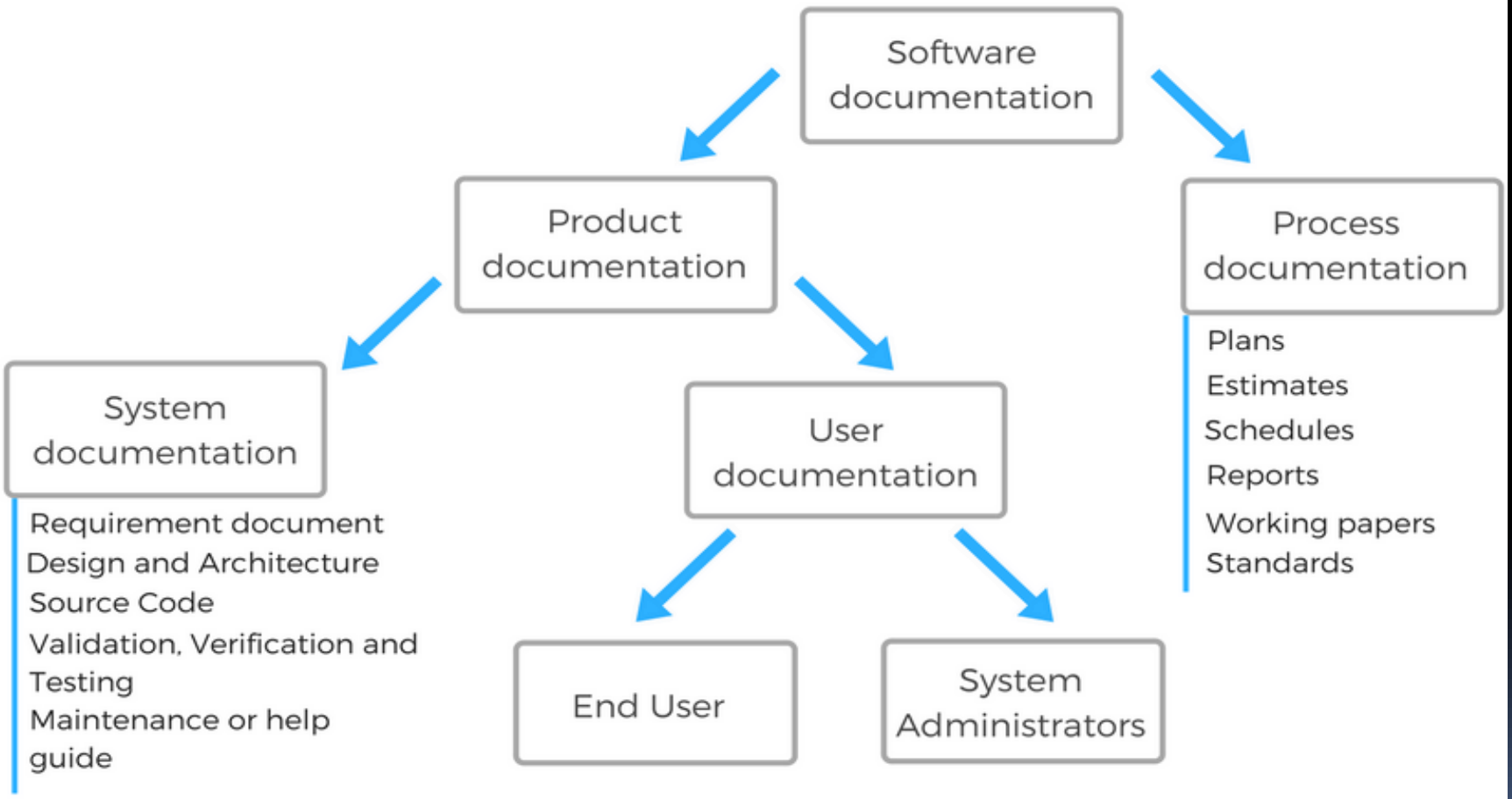
- Four key areas for testing a document include instructions, examples, messages, and samples.
- Instructions will be needed to step by step execute the test scenarios for looking errors or their omission.
- Further examples can be provided to elaborate the GUI components, syntax, commands and interfaces to show executed outputs or pin points.
- Inconsistencies also needed to be taken care of with errors as they can confuse the users and these ambiguities will cause much damage if the user of the system will be a novice user.

- Examples will be needed in case of any problem that occurs to the user, particularly novice users who may check the documentation for any confusion.
- Documentation problems can be handled in formal ways just the same way as the coding problems.[5] Defect reporting tools and tracking tools are the common solutions for handling defects just like as they are handled in code.

Importance of documentation testing

- Improves usability
  - Not all software was written for the Mac :-)
- Improves reliability
  - Testing the documentation is part of black-box testing.
  - A bug in the user manual is like a bug in the software.
- Lowers support cost
  - The exercise of writing the documentation helped debug the system.

Documentation Types



Domain testing approach using white box approach to test design

- **Domain testing** is a technique for **testing** software in which a minimum number of inputs are used to **test** the output of a system, to be sure that the system does not accept invalid and input values that are out of range. It is one of the essential **white box testing methods.**

1. Test adequacy criteria
2. Static testing
3. Structural testing
4. Code functional testing
5. Coverage and control flow graph

6.covering-code logic paths

7.Code-complexity testing

8.Evaluating test adequacy criteria.

- **What is Domain Testing?**

  Domain Testing is a type of Functional Testing which tests the application by giving inputs and evaluating its appropriate outputs.

In domain testing ,we divide a domain in to sub-domains(equivalence classes)and then test using values from each sub domain.

Ex: if a website (domain) has been given for testing , we will be dividing the website into small portions(sub domain)for the ease of testing.

It is a software testing technique in which the output of a system has to be tested with a minimum number of inputs in such a case to ensure that the system does not accept invalid and out of range input values.

- One of the most important White Box Testing method is a domain testing. The main goal of the Domain testing is to check whether the system accepts the input within the acceptable range and delivers the required output. Also, it verifies the system should not accept the inputs, conditions and indices outside the specified or valid range.

- **Adequacy criterion**: A **test adequacy criterion** is a predicate that is true (satisfied) or false (not satisfied) of a 〈program, **test** suite〉 pair. Usually a **test adequacy criterion** is expressed in the form of a rule for deriving a set of **test** obligations from another artifact, such as a program or specification.

- The goal for white box testing is to ensure that the internal components of a program are working properly. A common focus is on structural elements such as statements and branches.
- The tester develops test cases that exercise these structural elements to determine if defects exist in the program structure.
- The term exercise is used in this context to indicate that the target structural elements are executed when the test cases are run.
- By exercising all of the selected structural elements the tester hopes to improve the chances for detecting defects.

- Testers need a framework for deciding which structural elements to select as the focus of testing, for choosing the appropriate test data, and for deciding when the testing efforts are adequate enough to terminate the process with confidence that the software is working properly.
- Such a framework exists in the form of test adequacy criteria.
- Rules of this type can be used to determine whether or not sufficient testing has been carried out. The criteria can be viewed as representing minimal standards for testing a program. The application scope of adequacy criteria also includes:

(i)helping testers to select properties of a program to focus on during test;

(ii)helping testers to select a test data set for a program based on the selected properties;

(iii)supporting testers with the development of quantitative objectives for testing;

(iv)indicating to testers whether or not testing can be stopped for that program.

- If a test data adequacy criterion focuses on the structural properties of a program it is said to be a program-based adequacy criterion.
- Program-based adequacy criteria are commonly applied in white box testing.
- They use either logic and control structures, data flow, program text, or faults as the focal point of an adequacy evaluation.
- Other types of test data adequacy criteria focus on program specifications. These are called specification-based test data adequacy criteria. Finally, some test data adequacy criteria ignore both program structure and specification in the selection and evaluation of test data.
- Adequacy criteria are usually expressed as statements that depict the property, or feature of interest, and the conditions under which testing can be stopped (the criterion is satisfied).

The criteria can be viewed as representing minimal standards for testing a program. The application scope of adequacy criteria also includes:

- **helping testers to select properties of a program to focus on during test;**
- **helping testers to select a test data set for a program based on the selected properties;**
- **supporting testers with the development of quantitative objectives for testing;**
- **indicating to testers whether or not testing can be stopped for that program.**

- If a test data adequacy criterion focuses on the structural properties of a program it is said to be a program-based adequacy criterion. Program-based adequacy criteria are commonly applied in white box testing.
- They use either logic and control structures, data flow, program text, or faults as the focal point of an adequacy evaluation [1].
- Other types of test data adequacy criteria focus on program specifications.
- These are called specification-based test data adequacy criteria. Finally, some test data adequacy criteria ignore both program structure and specification in the selection and evaluation of test data.

Adequacy criteria are usually expressed as statements that depict the property, or feature of interest, and the conditions under which testing can be stopped (the criterion is satisfied). For example, an adequacy criterion that focuses on statement/branch properties is expressed as the following:

- **A test data set is statement, or branch, adequate if a test set T for program P causes all the statements, or branches, to be executed respectively.**

- In addition to statement/branch adequacy criteria as shown above, other types of program-based test data adequacy criteria are in use; for example, those based on (i) exercising program paths from entry to exit, and (ii) execution of specific path segments derived from data flow combinations such as definitions and uses of variables

- The concept of test data adequacy criteria, and the requirement that certain features or properties of the code are to be exercised by test cases, leads to an approach called coverage analysis, which in practice is used to set testing goals and to develop and evaluate test data.
- In the context of coverage analysis, testers often refer to test adequacy criteria as coverage criteria.
- For example, if a tester sets a goal for a unit specifying that the tests should be statement adequate, this goal is often expressed as a requirement for complete, or 100%, statement coverage. It follows from this requirement that the test cases developed must insure that all the statements in the unit are executed at least once.

- The nature of the unit

Some statements/branches may not be reachable.

The unit may be simple, and not mission, or safety, critical, and so complete coverage is thought to be unnecessary.

- The lack of resources
- The time set aside for testing is not adequate to achieve 100% coverage.
- There are not enough trained testers to achieve complete coverage for all of the units. There is a lack of tools to support complete coverage.
- • Other project-related issues such as timing, scheduling, and marketing constraints

Static testing:

- **Static testing** is software **testing** technique where **testing** is carried out without executing the code. This type of **testing** comes under Verification.
- In software development, **static testing**, also called dry run **testing**, is a form of software **testing** where the actual program or application is not used. Instead this **testing** method requires programmers to manually read their own code to find any errors. **Static testing** is a stage of White Box **Testing.**


Informal
Walkthrough
Peer Review
Inspection

- Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application.
- Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that can't not be found using Dynamic Testing, can be easily found by Static Testing.

The Static test techniques include:

- **Inspection**: Here the main purpose is to find defects. Code walkthroughs are conducted by moderator. It is a formal type of review where a checklist is prepared to review the work documents.
- **Walkthrough**: In this type of technique a meeting is lead by author to explain the product. Participants can ask questions and a scribe is assigned to make notes.
- **Technical reviews**: In this type of static testing a technical round of review is conducted to check if the code is made according to technical specifications and standards. Generally the test plans, test strategy and test scripts are reviewed here.
- **Informal reviews**: Static testing technique in which the document is reviewed informally and informal comments are provided

Dynamic Testing:

What is (Dynamic) structural Testing: Under **Dynamic Testing**, a code is executed. It checks for functional behavior of software system, memory/cpu usage and overall performance of the system. Hence the name "Dynamic".

The main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called an Execution technique or validation testing.

Dynamic testing is software **testing** technique where **testing** is carried out with executing the code. This type of **testing** comes under Validation.

- .

Dynamic Testing Techniques:

- **Unit Testing:** Under Unit Testing, individual units or modules are tested by the developers. It involves testing of source code by developers.
- **Integration Testing:** Individual modules are grouped together and tested by the developers. The purpose is to determine what modules are working as expected once they are integrated.
- **System Testing:** System Testing is performed on the whole system by checking whether the system or application meets the requirement specification document.
- **Acceptance Testing:** Testing done from user point of view at user's end.

Static Testing	Dynamic Testing
1. Static Testing is white box testing which is done at early stage of development life cycle. It is more cost effective than dynamic testing	1. Dynamic Testing on the other hand is done at the later stage of development lifecycle.
2. Static Testing Methods include Walkthroughs, code review.	2. Dynamic testing involves functional and nonfunctional testing
3. It is done before code deployment and without execution of code.	3. It is done after code deployment with the execution of codes.
4. This type of testing comes under Verification.	4. This type of testing comes under Validation.
5. Static testing achieves 100% statement coverage in a relatively short time.	5. Dynamic testing may involve running several test cases, each of which may take longer time.
6. Static testing is about prevention.	6. Dynamic testing is about cure.
7. In Static Testing techniques a checklist is prepared for testing process.	7. In Dynamic Testing technique the test cases are executed.

STATIC AND DYNAMIC TESTING

- **Difference between Static and Dynamic Testing:**

Static Testing	Dynamic Testing
Testing done without executing the program	Testing done by executing the program
This testing does verification process	Dynamic testing does validation process
Static testing is about prevention of defects	Dynamic testing is about finding and fixing the defects
Static testing gives assessment of code and documentation	Dynamic testing gives bugs/bottlenecks in the software system.
Static testing involves checklist and process to be followed	Dynamic testing involves test cases for execution
This testing can be performed before compilation	Dynamic testing is performed after compilation
Static testing covers the structural and statement coverage testing	Dynamic testing covers the executable file of the code
Cost of finding defects and fixing is less	Cost of finding and fixing defects is high

FUNCTIONAL TESTING

- **FUNCTIONAL TESTING** is a type of software **testing** that validates the software system against the **functional** requirements/specifications. ... **Functional testing** mainly involves black box **testing** and it is not concerned about the source **code** of the application.

- Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.

What do you test in Functional Testing?

- The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on -
- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

- **How to perform Functional Testing: Complete Process**
- In order to functionally test an application, the following steps must be observed.
- Understand the Software Engineering Requirements
- Identify test input (test data)
- Compute the expected outcomes with the selected test input values
- Execute test cases
- Comparison of actual and computed expected result

Examples of Functional testing are:

- Unit Testing
- Smoke Testing
- Sanity Testing
- Integration Testing
- White box testing
- Black Box testing
- User Acceptance testing
- Regression Testing

Functional Testing:

1. Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.
2. Functional testing is executed first.
3. Manual Testing or automation tools can be used for functional testing.
4. Business requirements are the inputs to functional testing.
5. Functional testing describes what the product does.
6. Easy to do Manual Testing.

Conclusion:

In Software Testing, Functional testing is a process of testing functionalities of the system and ensures that the system is working as per the functionalities specified in the business document. The goal of this testing is to check whether the system is functionally perfect!!!

Control flow graphs and code coverage

- One of the most important issue of software testing is that the testing process must cover the code under testing as much as possible.
- In software testing it is very important to use techniques and methods that find maximum faults in minimum time.
- Modern software became complex and contain large amount of source code so it is impossible to test all code statements manually, therefore there are many works and tools now for automatic software testing to reduce testing time and cost. ..

- There are many types of testing coverage metrics like statements coverage, conditional coverage, branch coverage, decision coverage, and function coverage . ...
- The application of coverage analysis is typically associated with the use of control and data flow models to represent program structural elements and data. The logic elements most commonly considered for coverage are based on the flow of control in a unit of code. For example,
-

- **(i) program statements;**
- **(ii) decisions/branches (these influence the program flow of control);**
- **(iii) conditions (expressions that evaluate to true/false, and do not contain** any other true/false-valued expressions);
- **(iv)combinations of decisions and conditions;**
- **(v) paths (node sequences in flow graphs).**

- These logical elements are rooted in the concept of a program prime. A program prime is an atomic programming unit. All structured programs can be built from three basic primes-sequential (e.g., assignment statements), decision (e.g., if/then/else statements), and iterative (e.g., while,for loops).

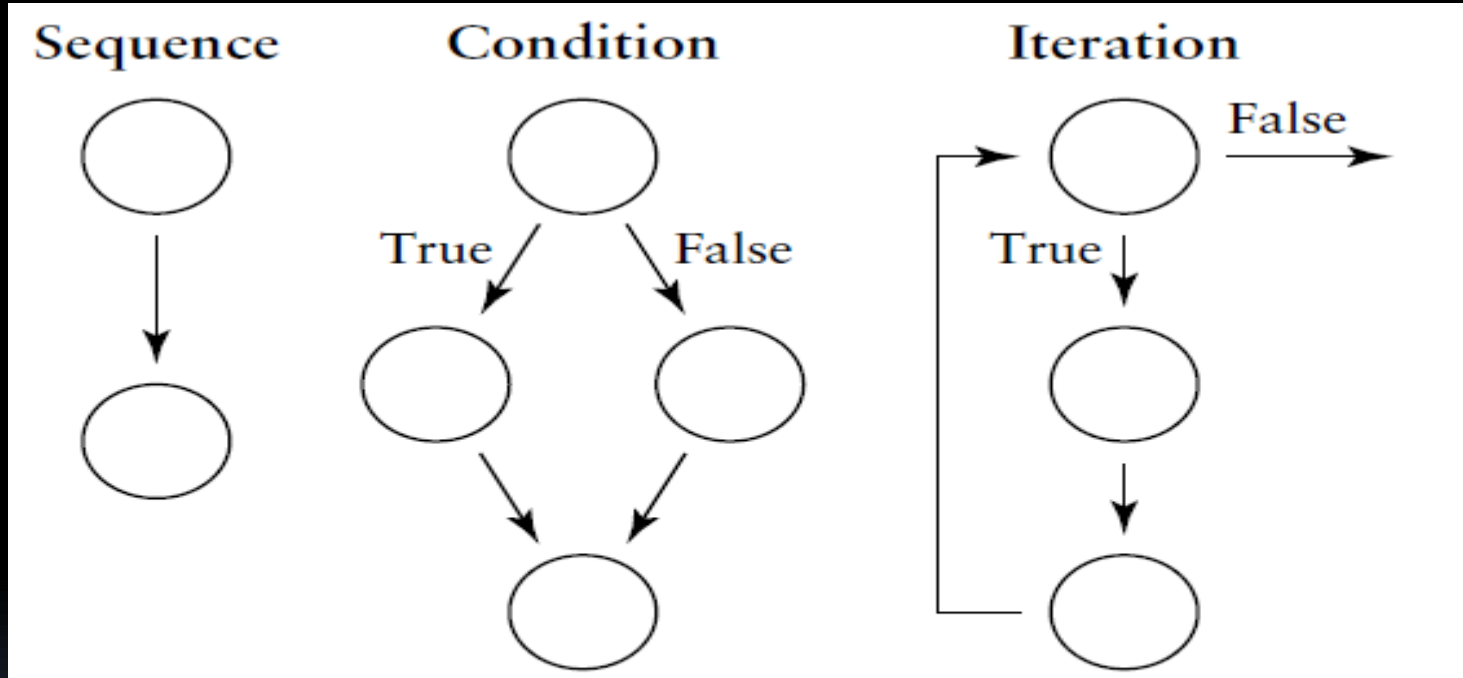


FIG. 5.1

Representation of program primes.

Covering Code Logic

- Logic-based white box-based test design and use of test data adequacy/ coverage concepts provide two major payoffs for the tester:
- (i) quantitative coverage goals can be proposed, and
- (ii) commercial tool support is readily available to facilitate the tester‘s work .

testers can use these concepts and tools to decide on the target logic elements (properties or features of the code) and the degree of coverage that makes sense in terms of the type of software, its mission or safety criticalness, and time and resources available.

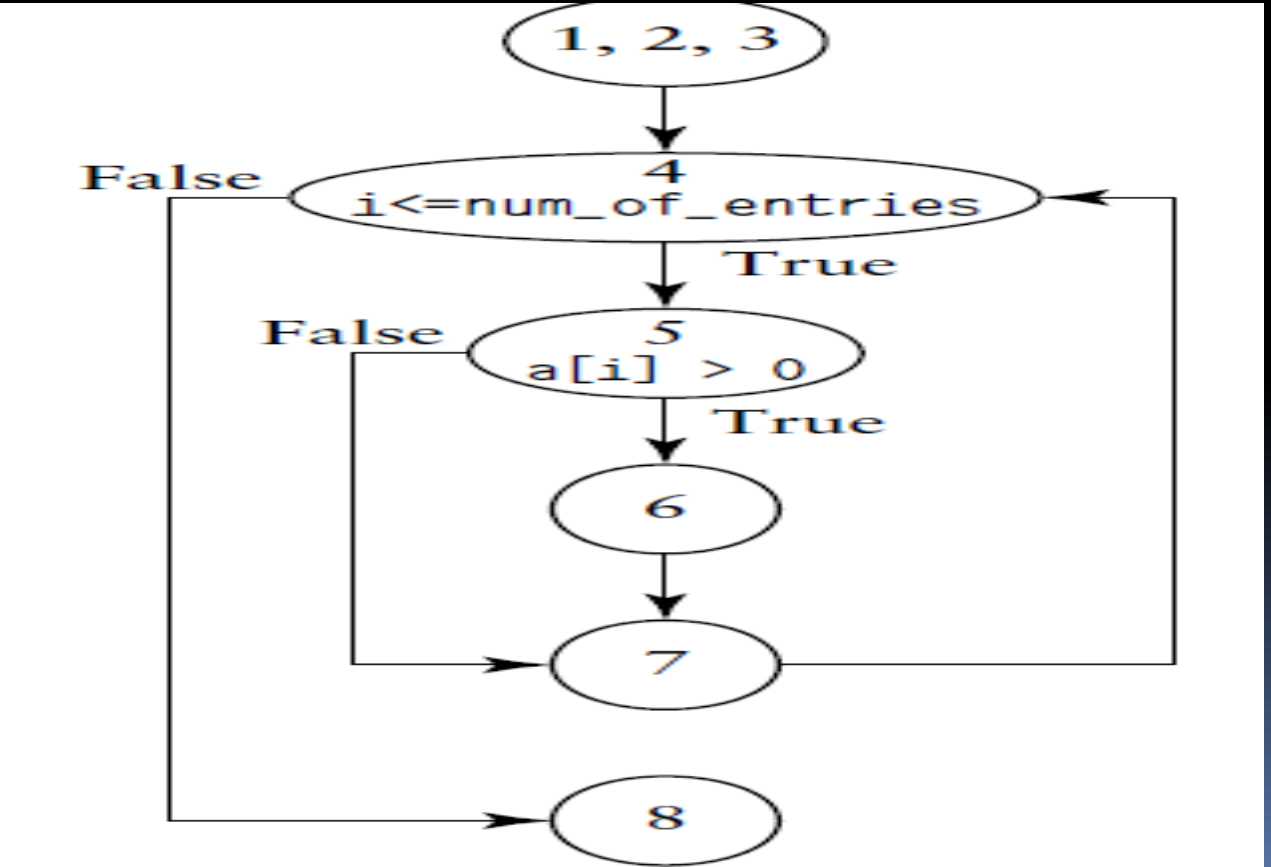
- For ex ample: if the tester selects the logic element program statements, this indicates that she will want to design tests that focus on the execution of program statements.
- If the goal is to satisfy the statement adequacy/ coverage criterion, then the tester should develop a set of test cases so that when the module is executed, all (100%) of the statements in the module are executed at least once.
- In terms of a flow graph model of the code, satisfying this criterion requires that all the nodes in the graph are exercised at least once by the test cases

- For the code in Figure 5.2 and its corresponding flow graph in Figure 5.3 a tester would have to develop test cases that exercise nodes 1-8 in the flow graph.
- If the tests achieve this goal, the test data would satisfy the statement adequacy criterion.
- In addition to statements, the other logic structures are also associated with corresponding adequacy/coverage criteria. For example, to achieve complete (100%) decision (branch) coverage test cases must be designed

- so that each decision element in the code (if-then, case, loop) executes with all possible outcomes at least once. In terms of the control flow model, this requires that all the edges in the corresponding flow graph must be exercised at least once.
- Complete decision coverage is considered to be a stronger coverage goal than statement coverage since its satisfaction results in satisfying statement coverage as well (covering all the edges in a flow graph will ensure coverage of the nodes)

- In fact, the statement coverage goal is so weak that it is not considered to be very useful for revealing defects. For example, if the defect is a missing statement it may remain undetected by tests satisfying complete statement coverage.
- he reader should be aware that in spite of the weakness, even this minimal coverage goal is not required in many test plans.
- Input values must ensure execution the true/false possibilities for the decisions in line 4 (while loop) and line 5 (if statement). Note that the if statement has a full else component, that is, there is no else part. However, we include a test that covers both the true and false conditions for the statement.

- Input values must ensure execution the true/false possibilities for the decisions in line 4 (while loop) and line 5 (if statement). Note that the if statement has a full else component, that is, there is no else part. However, we include a test that covers both the true and false conditions for the statement.



- A possible test case that satisfies 100% decision coverage .The reader should note that the test satisfies both the branch adequacy criterion and the statement adequacy criterion, since all the statements 1-8 would be executed by this test case.
- This code example represents a special case in that it was feasible to achieve both branch and statement coverage with one test case. Since one of the inputs, , is an array, it was possible to assign both positive and negative values to the elements of , thus allowing coverage of both the true/false branches of the if statement. Since more than one iteration of the while loop was also possible, both the true and false branches of this loop could also be covered by one test case. Finally, note that the code in the example does not contain any checks on the validity of the input parameters.

- This code example represents a special case in that it was feasible to achieve both branch and statement coverage with one test case. Since one of the inputs, , is an array, it was possible to assign both positive and negative values to the elements of , thus allowing coverage of both the true/false branches of the if statement
- . Since more than one iteration of the while loop was also possible, both the true and false branches of this loop could also be covered by one test case.
- Finally, note that the code in the example does not contain any checks on the validity of the input parameters. For simplicity it is assumed that the calling module does the checking.

What is Cyclomatic Complexity

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module .Cyclomatic Complexity is a very common buzz

word in the Development community.

This technique is mainly used to determine the complexity of a piece of code or functionality.

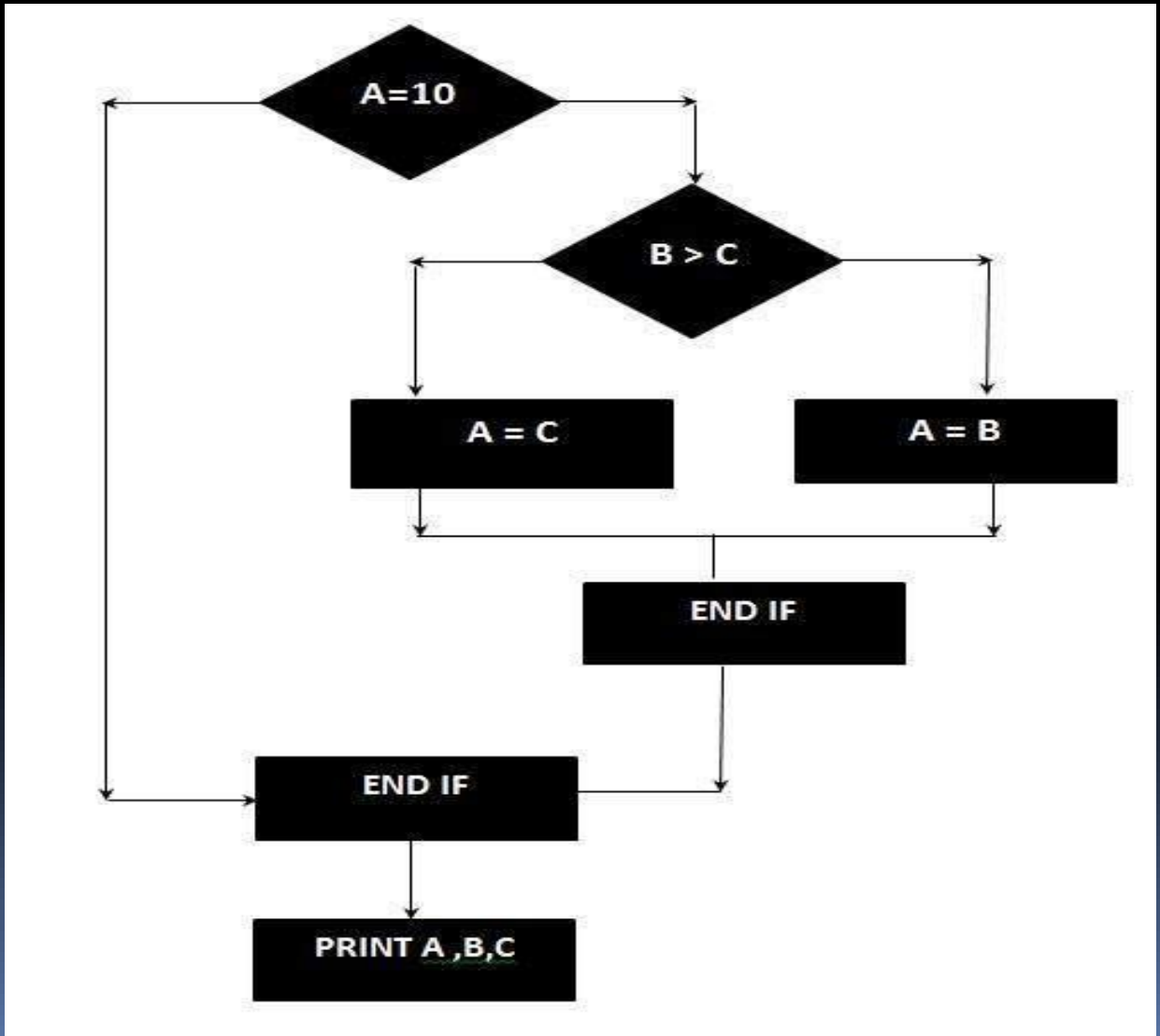
- The technique was developed by MaCabe and helps to identify the below 3 questions for the programs/features
- Is the feature/program testable?
- Is the feature/ program understood by everyone?
- Is the feature/program reliable enough

- Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:
- Cyclomatic complexity = E - N + 2*P where, E = number of edges in the flow graph. N = number of nodes in the flow graph. P = number of nodes that have exit points

Example :

- IF A = 10
- THEN IF B > C
- THEN A = B
- ELSE A = C
- ENDIF
- ENDIF
- Print A Print B Print C

Flow graph:



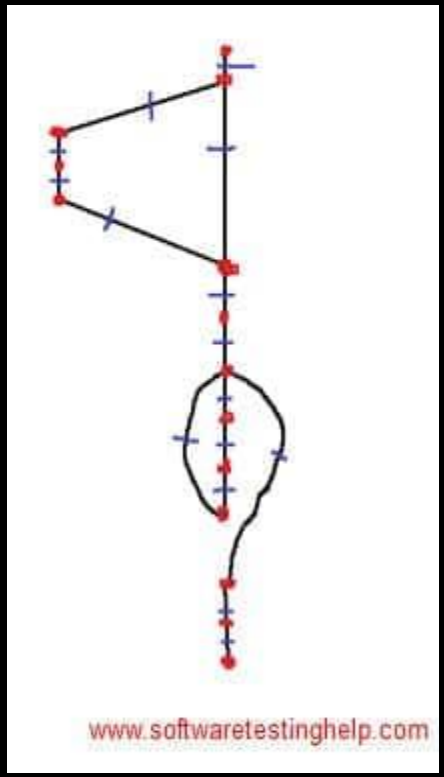
- The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes(shapes) and eight edges (lines), hence the cyclomatic complexity is 8 - 7 + 2 = 3

- **Let me go step by step:** first understand how is it calculated, and then we will move on to understand how the level of testing is determined.
- **How to Calculate Cyclomatic Complexity?**
- **The calculation of CC revolves around 2 concepts**
- Nodes
- Edges
- Statements in a program are represented as nodes, and control paths from one statement to another are represented by Edges.
- **Cyclomatic Complexity formula**
- **The formula for calculating CC is as:**
- CC = E~N+2

Where:

- E= Number of edges
- N= Number of nodes.
- *(There is a shortcut for calculating it, but not now……later…)*
- **Cyclomatic Complexity Example**
- Let us take the below example to understand it.
- Consider the below Control flow graph:

www.softwaretestinghelp.com



I have placed the **RED** dots to identify the Nodes and **BLUE** lines to identify the edges:

So here in this example:

- Number of Nodes (Red dots) = 14
- Number of Edges (Blue Lines) = 15
- So the Cyclomatic Complexity = N~E+2 = (14-15) +2 = 3

How can Testers use it?

- In real-world, Testers can sit with developers to derive the control flow graph for a given piece of code. And once we have the graph, we can derive the complexity using this formula. But the story for Testers does not end here: – the main point here is – what is the use of this number for the testing team?
- Well, Testers can make use of this number to determine the level of their testing.

In practice there are 2 levels of testing:

- Length Testing
- Breadth Testing

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
Sub Feature 1.1	Sub Feature 2.1	Sub Feature 3.1	Sub Feature 4.1	Sub Feature 5.1
Sub Feature 1.2	Sub Feature 2.2	Sub Feature 3.2	Sub Feature 4.2	Sub Feature 5.2
Sub Feature 1.3	Sub Feature 2.3	Sub Feature 3.3	Sub Feature 4.3	Sub Feature 5.3
.....	.....	.....	.....	.....
.....	.....	.....	.....	.....

So based on your current project requirement, environment dependability, testers can collaborate together with the development team and create a standard for identification of the level and scope of testing. For example –

- If the CC <=15 – Basic sanity test
- If the CC is between 16 and 30 – Length Testing
- If the CC is between 31 and 50 – Breadth testing
- If the CC >50 – It's a chaotic functionality and needs further decomposition.
- **Now comes the shortcut-**
- *Just count the number of the closed regions and add 1 to it.*
- *In our example above – number of closed region = 2(filled in yellow), so the CC = 2+1 = 3*