

# **Programming Foundation With Pseudocode**

Lesson 5: Software Reviews  
and Testing

## Lesson Objectives

- To Understand the following concepts
  - What is software Testing?
  - What is Debugging?
  - Software Testing Principles
  - TestCase
  - Exhaustive Testing
  - Testing Techniques
  - Static Testing
  - Dynamic Testing
  - Testing Approaches
  - Unit Testing
  - Integration Testing
  - System Testing
    - Verification and Validation testing
    - Acceptance Testing
    - Regression testing



5.1. What is Software Testing?

## What is Software Testing?

- Testing is the process of executing a program with the intent of finding errors
- Testing is a process used to help identify the correctness, completeness and quality of a developed computer software
- Testing helps in Verifying and Validating if the Software is working as it is intended to be working



Copyright © Capgemini 2015. All Rights Reserved 3

### What is software testing?

Exercising (analyzing) a system or component with

defined inputs

capturing monitored outputs

comparing outputs with specified or intended requirements

To maximize the number of errors found by a finite no of test cases.

Testing is successful if you can prove that the product does what it should not do and does not do what it should do.

5.1. What is Software Testing?

## Successful test

- What is a successful Test?
  - If we run all tests on a program, and we do not find any defects, what is the conclusion?
  - “The program quality was good as it passed all tests”. Is it?
- OR
- “The testing quality was poor as it failed to find any defects”



Copyright © Capgemini 2015. All Rights Reserved 4

Testing is in a way a destructive process and a successful test case is one that brings out an error in the program . Detection of an error/failure is a success

5.2. What is Debugging?

## Debugging

### ■ Debugging

- Is an art used to “isolate”, and “correct” the cause of an error
- Debugging can be performed on code or on requirements and specifications.
- Debugging is performed by developers to uncover where a defect in the code exists and correct it.
- Combines a “systematic search” with an intuitive feel for the nature of the program
- May take an hour, day, or a month. Hence difficult to reliably schedule



Copyright © Capgemini 2015. All Rights Reserved 5

Objective of debugging is to find and correct the cause of the software error. Debugging is not testing.

The symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed.

The symptom may be caused by round off in accuracies.

Intermittent problems in embedded systems because hardware is tightly coupled with software

As the consequences of an error increase, the amount of pressure to find the cause also increases. Often, pressure forces a software developer to fix one error while at the same time introducing two more.

The debugging process has one of the two outcomes:

The cause will be found, corrected and removed.

The cause will not be found.

In the second outcome, the person performing debugging may suspect a cause, design a test case to help validate his or her suspicion and work toward error correction in iterative manner

5.2. What is Debugging?

## Debugging Techniques

- Debugging can be done by using the following subtypes:
  - Brute Force
    - Storage Dump
    - Scattering Display Statements
    - Run time Traces
  - Backtracking Method
- Backtrack the incorrect results
  - Cause Elimination
  - Proceeding from some general theories



Copyright © Capgemini 2015. All Rights Reserved 6

**Brute Force:** It is the most common and least efficient method for isolating the cause of a software error. We apply brute force debugging methods when all else fails.

Example of debugging by Brute Force are

1. By studying Storage Dumps I.e. usually a crude display of storage location
2. by invoking run-time traces
3. by scattering print statements
4. by use of automated debugging tools

**Backtracking** is a common debugging approach. It is basically used in small programs. The program code is manually tracked beginning at the place where the symptom is uncovered, the source code is traced backward until the site of the cause is found.

### Cause elimination

Debugging by Induction

1. Locate data about what program did correctly/incorrectly
2. Organize data
3. Devise a hypothesis about the cause of the error
4. Prove the hypothesis

Debugging by deduction

1. Enumerate the causes of error
2. Eliminate each cause of error

5.2. What is Debugging?

## Comparison

- Purpose of Testing
  - To show that a program has a bug
- Purpose of Debugging
  - To find and correct the cause of an error

## 5.3 Testing Principles

## Testing Principles

- A necessary part of a test case is a definition of the “expected output” or “result”
- Test cases must be written for “invalid and unexpected” as well as “valid and expected” input conditions
- The probability of finding “more defects” in a module is proportional to the “number of defects already found” in that module
- In most systems, 20% of the modules account for 80% of the defects found
- A programmer should not be the only person to test his or her own program



5.4: Test Case

## What is Test Case?

- “A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement”
- In other words, a planned sequence of actions (with the objective of finding errors)
- Test cases may be designed based on
  - Values – Valid/Invalid/Boundary/Negative
  - Test conditions



Copyright © Capgemini 2015. All Rights Reserved 9

A Test case is a planned sequence of actions.

Characteristics of a Good Test:

They are: likely to catch bugs  
not redundant  
not too simple or too complex.

5.4: Test Case

## Example

- Problem:

- Given the lengths of three sides of a triangle, determine whether a valid triangle is formed.
- If valid, determine the type of triangle – equilateral, isosceles, or scalene.
- Develop the code for the above problem.
- Identify all the test cases required to test the code, by using the following headers:
  - Test Case #, Test Case description, Expected result



Copyright © Capgemini 2015. All Rights Reserved 10

While testing the code consider valid expected and invalid unexpected scenarios.

5.4: Test Case

## Test cases - Example

### Valid Test cases:

▪ Scalene	3,4,5;	4,5,3;	5,3,4;
▪ Isosceles	3,4,4;	4,3,4;	4,4,3;
▪ Equilateral	3,3,3;	4,4,4	5,5,5

### Invalid test cases

▪ Scalene	3,3,a	3,4,-1	1,2,0
▪ Isosceles	3,-1,3	1,2,3	3,4,0
▪ Equilateral	0,0,0;	-1,-1,-1	



Copyright © Capgemini 2015. All Rights Reserved 11

If you observe the examples given on the slide, Some of the input values are checking for valid values for scalene, isosceles, equilateral triangle. and some of the input values are checking invalid values like

- Length of a triangle cannot be negative
- Length of triangle cannot be zero
- Length of a triangle cannot be alphabet etc

5.4: Test Case

## How to write Test cases

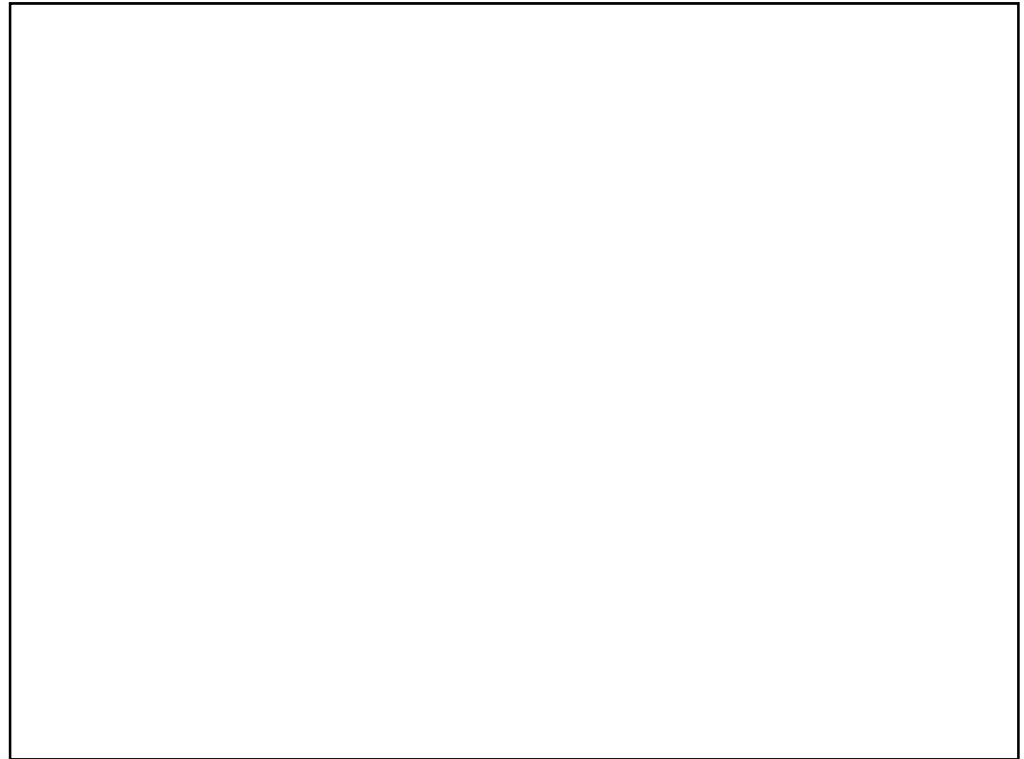
- Write test case
  - for both valid and invalid values
  - to test all fields separately as well as field boundaries
  - to test form submission and URL navigation
  - to detect high probability of errors
  - to maximize bug count
  - to assess conformance to specification
  - to verify correctness of the product
  - to assess quality



Copyright © Capgemini 2015. All Rights Reserved 12

### How to write test case?

- Write test case for both valid and invalid values. For an Example, if you want to validate the salary, then consider the below mentioned test cases.
  - write test case with valid input data as "10000".
  - write test case with invalid input data as "Test".
- Write test case to test all fields separately as well as field boundaries. For an Example, consider a login form contains two fields like username and password. Write both valid and invalid test cases for all the fields(i.e, Username and Password) available in the page.
- Write test case to test form submission and URL navigation. For an Example, consider a login form contains two fields like username and password. If you want to navigate to the next page, while clicking on submit button after entering valid username and password, then prefer writing test case for form submission.
- Defect high probability of errors: This is the classic objective of testing. A test is run in order to trigger failures that expose defects. Generally, we look for defects in all interesting parts of the product.
- Maximize bug count: The distinction between this and “find defects” is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high-risk features, if this is the way to find the most bugs in the time available.



- Assess conformance to specification. Any claim made in the specification is checked. Program characteristics not addressed in the specification are not (as part of this objective) checked.
- Verify correctness of the product. It is impossible to do this by testing. You can prove that the product is not correct or you can demonstrate that you didn't find any errors in a given period of time using a given testing strategy. However, you can't test exhaustively, and the product might fail under conditions that you did not test. The best you can do (if you have a solid, credible model) is assessment--test-based estimation of the probability of errors. (See the discussion of reliability, above).
- Assure quality. Despite the common title, quality assurance, you can't assure quality by testing. You can't assure quality by gathering metrics. You can't assure quality by setting standards. Quality assurance involves building a high quality product and for that, you need skilled people throughout development who have time and motivation and an appropriate balance of direction and creative freedom. This is out of scope for a test organization. It is within scope for the project manager and associated executives. The test organization can certainly help in this process by performing a wide range of technical investigations, but those investigations are not quality assurance. Given a testing objective, the good test series provides information directly relevant to that objective. Different types of tests are more effective for different classes of information.

5.4: Test Case

## How to write Test cases

- A test case may contain the following fields
  - Requirement Id
  - Test Case Id
  - Test condition
  - Test cases
  - Test data
  - Expected result
  - Remarks



Copyright © Capgemini 2015. All Rights Reserved 14

Test case has to be written to validate the testing coverage of the application.

**Requirement Id:** The ID of the requirement this test case relates/traces to.

**Test Case Id:** Unique ID for each test case. Follow some convention to indicate types of test. E.g. 'TC\_UI\_1' indicating 'user interface test case #1'.

**Test condition:** Describes what to test for i.e. the condition which is being tested for example, Validate name

**Test cases:** Step-by-step procedure to execute the test.

**Test data:** Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input

**Expected result:** What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on screen

**Remarks:** Any comments on the test case or test execution.

5.4: Test Case

## Guidelines for implementing test cases

- Test if all the requirements are covered in the application.
- Don't miss out to test non functional requirements if mentioned in the requirement.
- Raise defect for all test cases that fail.
- Errors may creep in in boundaries, so check for all boundary conditions.
- Don't forget 80-20 rule.
- Quality of test case affect testing, so review all test cases.
- Self review your test cases as quality of test case affects testing.



Copyright © Capgemini 2015. All Rights Reserved 15

### Guidelines for implementing test cases:

- Write test case for all the requirements specified in the application
- Take care of writing test case for non functional requirements like security, performance, etc..
- If any test case fails, log the failed test cases as defect in defect tracking sheet.
- Check for all boundary conditions.
- **80-20 Rule:** In most systems, 20% of the modules account for 80% of the defects found. The probability of finding defect in a module is directly proportional to the number of defects already found in the module.
- Do self review and peer review for all test cases as quality of test case affects testing.

## Demo : Test Case creation

- Test case example



Refer Hotel Bookings Management System.doc and Test\_Case-HotelBookingsSystem.xls file to understand test case example.



## 5.5 Exhaustive Testing

## Exhaustive and Economics of testing

### ■ Exhaustive Testing

- Exhaustive Testing involves testing for every possible input, and every possible output
- For example: Online railway reservation system
  - It is impossible to test the ticket booking for all possible combination of sources and destinations
  - Hence we test the code with some sample values
- However, "Exhaustive Testing" is impractical, and not economically viable
- Therefore, the objective of testing is to find "maximum errors" with a finite number of test cases

### ■ Economics of Testing

- It is both the driving force and the limiting factor
- Driving - Earlier the errors are discovered and removed in the lifecycle, lower the cost of their removal.
- Limiting - It is infeasible to test exhaustively all possible combinations.



Copyright © Capgemini 2015. All Rights Reserved 17

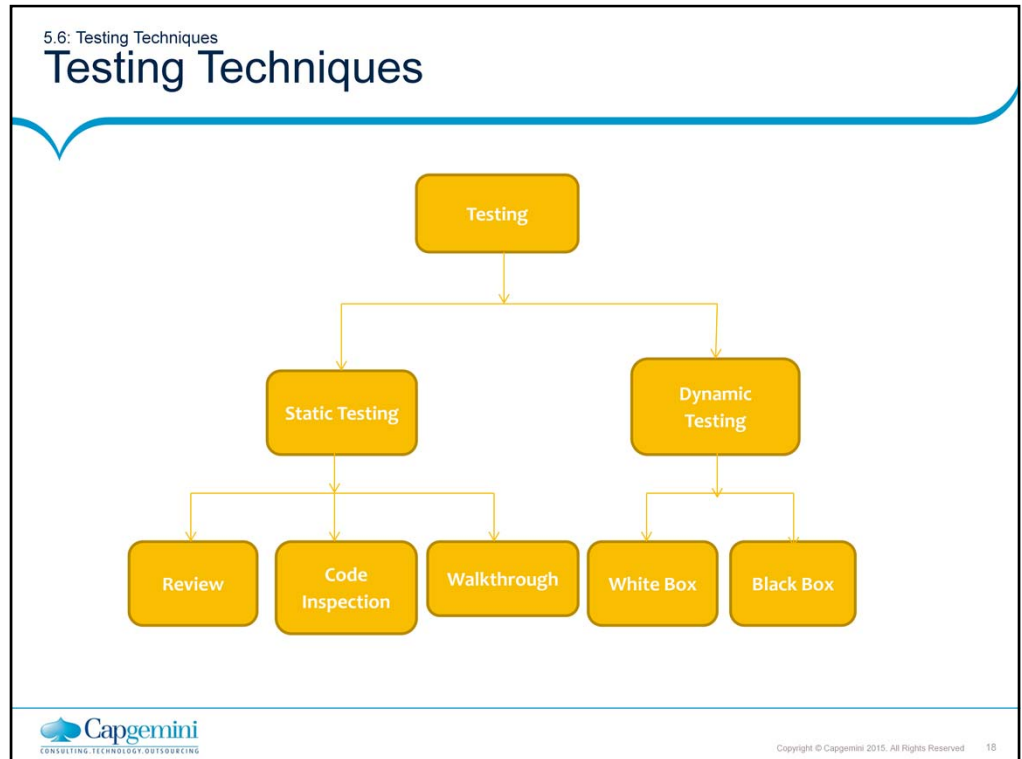
Exhaustive testing is impossible.

For E.g. COBOL Compiler

- Impossible to create test cases to represent all valid cases.
- Impossible to create test cases for all invalid cases

The compiler has to be tested to see that it does not do what it is not supposed to do E.g. to successfully compile a syntactically incorrect program

But writing Cobol programs to include all possible syntactical errors that compiler check is impractical. It is time consuming and hence not suggestible.



- Static Testing: Testing a software without execution on a computer
  - Review : Review the created artifacts using checklist
  - Code Inspection : Code inspection is a set of procedures and error detection techniques for group code reading.
  - Walkthrough : Like code inspection it is also an group activity.
- Dynamic Testing techniques exercise the software by using sample input values
  - WhiteBox testing : Used to test the internal structure of the code
  - BlackBox Testing : Test the functionality of application by providing input and getting expected output

## 5.7. Static Testing

## Definition of static Testing

- Static Testing is a process of reviewing the work product using a checklist
- Testing a software without execution on a computer
- Involves just examination/review and evaluation
- Use “Static Analysis” or “Static Testing” for
  - Examining “Control flow” and “Data flow”
  - Discovering dead code, infinite loops, un-initialized and unused variables, standard violations, etc.
  - Finding 30 - 70% of errors effectively



Copyright © Capgemini 2015. All Rights Reserved 19

### Static Testing:

- Static Testing is a process of reviewing the work product using a checklist.
- No need to execute a program for performing static Testing.
- Static testing may be conducted manually or through the use of various software review tools like PMD, Checkstyle. Specific types of static software testing include code analysis, inspection, code reviews and walkthroughs.
- Through static testing, errors in the control flow/data flow can be identified at the earlier stage.

## 5.7: Static Testing

## Static Testing Methods

### ■ Static Testing Methods

#### ■ Self Review

- Done by the author himself with the aid of tools like checklists, review guidelines, rules, etc

#### ■ Code Inspection

- It is a more systematic and rigorous type of peer review.
- Code inspection is a set of procedures and error detection techniques for group code reading.
- Involves reading or visual inspection of a program by a team of people, hence it is a group activity

#### ■ Walk Through

- Like code inspection it is also an group activity.
- In Walkthrough meeting, three to five people are involved. Out of the three, one is moderator, the second one is Secretary who is responsible for recording all the errors and the third person plays a role of Test Engineer.



Copyright © Capgemini 2015. All Rights Reserved 20

### Review Process

**Input :** Work Product, Specifications, Checklists, Guidelines, Historical Data

#### **Process**

- Prepare for Review
- Conduct Reviews
- Analyze Deviations
- Correct Defects

**Output :** Review Form, reviewed work product

5.8. Dynamic Testing

## Dynamic Testing

- Dynamic Testing techniques exercise the software by using sample input values
- Dynamic Testing is classified as:
  - Functional Test Case Selection Technique (or Black Box Testing)
    - Test the functionality of application by providing input and getting expected output
  - Structural Test Case Selection Technique (or White Box Testing)
    - Used to test the internal structure of the code



Copyright © Capgemini 2015. All Rights Reserved 21

### Dynamic Testing

- Dynamic Testing involves the testing of a software by executing the system.
- Using Dynamic Testing, internal structure and functionality of an application will be tested.
- Perform white box testing for testing the internal structure of the code
- Test the functionality of an application using Black box testing.

For an Example, if you want to validate all the fields in the login page to accept valid data, then perform black box testing.

Steps to be performed for black box testing are:

1. Execute an application
2. Type the input
3. Validate the actual result against the expected result mentioned in the test plan.
4. If actual result and expected result matches, then the test case result is pass else the result is fail.

For an Example, if you want to check whether the logic(code) of login functionality is working fine, then perform white box testing.

## 5.8 Black Box Testing

## Black Box Testing: Features and Techniques

- Black Box Testing has following characteristics:
  - The internal structure of the code is not tested
  - Main focus is on testing whether the input is properly accepted, and output is correctly produced
  - The integrity of external information (data files) is maintained
- Black Box Testing comprises of the following techniques:
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Error Guessing



Copyright © Capgemini 2015. All Rights Reserved 22

E.G

While testing washing machine, you need not be aware of internal structure of washing machine.


We need to know how to use interface and give instructions to washing machine.

Hence in black box testing we need to know what is input and what is output.

5.9 Testing Approaches

## Testing Approaches

- Testing Approaches are
  - Unit testing
  - Integration testing
  - Validation testing
  - System testing
  - Acceptance testing
  - Regression testing

 Capgemini  
EMERGING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

### Testing Approaches:

- Unit testing is code-based and performed primarily by developers to demonstrate that their smallest pieces of code execution works properly.
- Integration testing demonstrates that two or more units or other integrations work together properly.
- Validation Testing can be used for performing validation of software typically includes evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements.
- System testing demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design(both functional and not functional requirement)
- Acceptance testing is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.
- Regression Testing is the testing of software after a modification has been made to ensure the reliability of each software release.

5.9.1 Unit Testing

## Unit Testing

- Module Testing
- Done by Programmers
- Discover discrepancies between the unit's specification and its actual behavior
- Testing a form, a component or a stored procedure can be an example of unit testing



Copyright © Capgemini 2015. All Rights Reserved 24

What is a Unit?

Synonyms are “component” and “module.”

The IEEE glossary says (for module):

- A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.
- A logically separable part of a program.

Unit Testing:

- The most 'micro' scale of testing to test particular functions, procedures or code modules. Also called as Module testing.
- Typically done by the programmer and not by Test Engineers, as it requires detailed knowledge of the internal program design and code.
- Purpose is to discover discrepancies between the unit's specification and its actual behavior.
- Testing a form, a class or a stored procedure can be an example of unit testing



## 5.9.2 Integration Testing

## Integration Testing

- Integration Testing focuses on testing a combination of two or more modules
- The different Integration Testing strategies are:
  - Big Bang approach
  - Incremental approach
    - Top-Down approach
    - Bottom-Up approach
    - Sandwich approach
- The terminologies related to Integration Testing
  - Stub:
    - Simulation of a subordinate module
  - Driver:
    - Simulation of a super-ordinate module



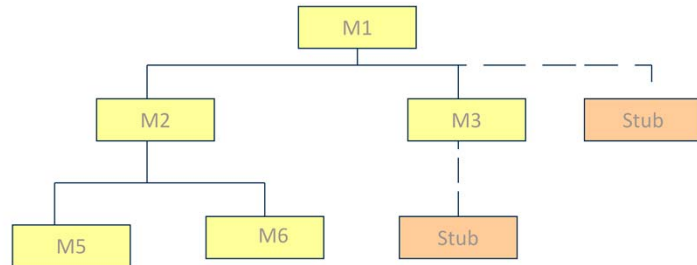
### Non-incremental Testing (Big Bang Testing)

Each Module is tested independently and at the end, all modules are combined to form a application.

## 5.9.2 Integration Testing

## Top Down Integration Testing

- Top Down Incremental Module Integration:
  - Topmost module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested.



Copyright © Capgemini 2015. All Rights Reserved 26

The main control module is used as a test driver. Stubs are substituted for all components directly subordinate to the main control module. Depending on the approach subordinate stubs are replaced by actual components.

**Disadvantages:**

Many tests are delayed until stubs are replaced by actual modules.

Time taken to develop stubs to perform the functions of the actual modules.

**Advantage :**

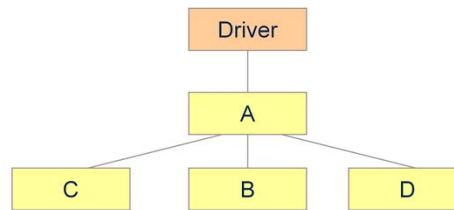
Fast

5.9.2 : Integration testing

## Bottom Up Integration Testing

- Bottom Up Incremental Module Integration:

- Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested. This continues till top most module is added to rest all and tested



Low-level components are combined into clusters (builds) that perform a specific sub function. A driver is written to coordinate test case input and output. Drivers are removed and clusters are combined moving upward in the program structure.

5.9.3 System Testing

## What is System Testing?

- System Testing focuses on:
  - a complete integrated system as a whole, in order to evaluate compliance with respect to specified requirements
  - characteristics that are present only when the entire system is up and running
  - series of different tests, whose primary purpose is to verify that all system elements are properly integrated, and are performing allocated functions
  - Two types of System Testing
    - Functional
    - Functional Testing will be performed to validate if the output is correct for the given input.
  - Non Functional
    - Non Functional Testing will be used to check the other important aspects of an application like security, performance and usability.



Copyright © Capgemini 2015. All Rights Reserved 28

### Types of System Testing:

- Functional Testing will be performed to validate if the output is correct for the given input.
- Non Functional Testing will be used to check the other important aspects of an application like security, performance and usability.

5.9.4 Verification and Validation Testing

## Validation Testing

- Purpose:
  - to show that the program does not match it's external specifications
  - to have a final check to see whether it is indeed the right product
- Verification:
  - Is the product error-free? Is it as per the product specifications?
- Validation:
  - Is it fit for use? Are end-users satisfied?

## 5.9.5 Acceptance Testing

## Acceptance Testing

- Acceptance Testing focuses on testing whether the right system has been created. It is usually carried out by the end user
- The two types of Acceptance Testing are:
  - Alpha testing
    - Generally, done at the developer's site in the presence of the developer.
  - Beta testing
    - Done at the customer's site with no developer at the site.



Copyright © Capgemini 2015. All Rights Reserved 30

### Acceptance Testing:

A test executed by the end user(s) in an environment simulating the operational environment to the greatest possible extent, that should demonstrate that the developed system meets the functional and quality requirements.

5.9.6 Regression Testing

## Regression testing?

- Regression Testing involves “selective re-testing” of the system or it’s components after the changes are done
- Regression Testing is done to:
  - verify absence of unintended effects
  - verify compliance with all (old and new) requirements
- The Regression Testing strategy involves:
  - running new test cases for the newly introduced modules
  - re-running old test cases to check the effect on unchanged modules



Copyright © Capgemini 2015. All Rights Reserved 31

### Regression Testing:

- Regression Testing is the testing of software after a modification has been made to ensure the reliability of each software release.
- Testing after changes have been made to ensure that changes did not introduce any new errors into the system.
- It applies to systems in production undergoing change as well as to systems under development
- Re-execution of some subset of test that have already been conducted is required

## Lab

- Review and Software Testing Lab Exercises –
  - Lab 5.1 and 5.2



Do review of existing pseudocode and write test case to perform unit testing of an program which you have created.



## Lesson Summary

- In this lesson, you have learnt about
  - Testing is a process to identify errors
  - A successful test case is one that brings out an error in the program
  - Exhaustive testing
  - Testing Techniques like black box and white box.
  - Testing approaches like
    - Unit Testing
    - Integration Testing
    - Verification and Validation Testing
    - System Testing
    - Acceptance Testing
    - Regression Testing



## Review - Questions

### Question 1: Alpha Testing is -----

- A : done at the developer's site in the presence of the end user.
- B: done at the developer's site in the presence of the developer.
- C: done at the end user's site in the presence of the developer.
- D: done at the end user's site in the presence of the end user.



### Question 2 : Which of the following are true for testing

- A: Test cases should be designed for valid and expected values only
- B: While testing you should check for each and every possible value
- C: White Box Testing focuses on the internal structure of the software
- D: The purpose of testing is to find and correct the cause of the error

## Review - Questions

- Question 3 : In top down approach of integration testing we replace modules with -----
  - A: driver
  - B: stub
  - C: control module
  - D: All of the above



## Review – Match the Following

■ Question 4 :

1. Acceptance testing	A. Big Bang approach
2. Regression testing	B. Exhaustive testing
3. Integration testing	C. Beta testing
4. Use every possible input condition as a test case	D. Selective retesting
5. Debugging	E. Maximize bug count
	F. Storage dump

