

# Testing Concepts for V&V Automation Testing

Lesson 0

People matter, results count.



## Document History

| Date       | Course Version No. | Software Version No. | Developer / SME | Reviewer(s) | Approver      | Change Record Remarks                       |
|------------|--------------------|----------------------|-----------------|-------------|---------------|---|
|            | 1.0                | MS Office 2003       |                 |             |               | Content Creation                            |
| May-2009   | 1.0                | MS Office 2007       | Priya Rane      |             |               | Material Revamp                             |
| June-2011  | 1.1                | MS Office 2007       | Hema G.         |             |               | Material Revamp                             |
| April-2014 | 2.0                | MS Office 2010       | Dayanand Patil  | Samir Attar | Shilpa Bhosle | Material Revamp                             |
| Dec-2015   | 2.1                | MS Office 2010       | Shilpa Bhosle   | Samir Attar | Shilpa Bhosle | Customized for V&V – Automation Testing LoT |



Copyright © Capgemini 2015. All Rights Reserved 2

## Course Goals and Non Goals

- Course Goals

- At the end of this program, participants gain an understanding of how to create effective test cases using the different testing techniques to get a good test coverage of a software application.

- Course Non Goals

- This course does not cover automation process of testing.



Copyright © Capgemini 2015. All Rights Reserved 3

### Pre-requisites

- None



Copyright © Capgemini 2015. All Rights Reserved 4

## Intended Audience

- Test Engineers, Software Engineers and Senior Software Engineers.



Copyright © Capgemini 2015. All Rights Reserved 5

## Day Wise Schedule

- Day 1

- Lesson 1: Introduction to Software Testing
  - Lesson 2: Types of Testing Techniques & Test Case Design

- Day 2

- Lesson 2: Types of Testing Techniques & Test Case Design (Cont.)
  - Lesson 3: Types of System Testing

- Day 3

- Lesson 3: Types of System Testing (Cont.)
  - Lesson 4: Test Case Execution / Testing Metrics



Copyright © Capgemini 2015. All Rights Reserved 6

## Table of Contents

- Lesson 1: Introduction to Software Testing
  - 1.1 Testing Overview
  - 1.2 What is testing
  - 1.3 Why testing
  - 1.4 How testing
  - 1.5 Testing principles
  - 1.6 Objective of testing
  - 1.7 Exhaustive testing
  - 1.8 Limitations of software Testing
  - 1.9 Psychology of testing
  - 1.10 Test case
  - 1.11 Test case formats
  - 1.12 Other related terminologies



Copyright © Capgemini 2015. All Rights Reserved 7

## Table of Contents

- Lesson 2: Types of Testing Techniques & Test Case Design
  - 2.1. Types of testing techniques- Static, Dynamic
  - 2.2. Test Case Construction / Test Data Preparation
    - 2.2.1. Introduction
    - 2.2.2. Test Case
    - 2.2.3. Test case life cycle
  - 2.2.3.1. Introduction
  - 2.2.3.2. Test Case Design techniques
    - 2.2.3.2.1. Black Box testing – Boundary Value Analysis, Equivalence Partitioning, Decision Table
    - 2.2.3.3. Conventional Test Design Approach
    - 2.2.3.3.1. Complexity Based
  - 2.3. Positive Tests
    - 2.3.1. What is positive testing
    - 2.3.2. Example of positive testing
    - 2.3.3. Advantages and Limitations of positive testing
    - 2.3.4. Positive test scenarios



Copyright © Capgemini 2015. All Rights Reserved 8

## Table of Contents

- Lesson 2: Types of Testing Techniques & Test Case Design
  - 2.4. Negative tests
    - 2.4.1. What is negative testing
    - 2.4.2. Example of negative testing
    - 2.4.3. Advantages and Limitations of negative testing
    - 2.4.4. Negative test scenarios
  - 2.5. Basic Tests
    - 2.5.1. What is basic test
    - 2.5.2. Example of basic test
  - 2.6. Alternate Tests
    - 2.6.1. What is alternate test
    - 2.6.2 Example on alternate test
  - 2.7. Importance of writing positive,negative,basic,alternate test while designing test cases



Copyright © Capgemini 2015. All Rights Reserved 9

## Table of Contents

- Lesson 3: Types of System Testing
  - 3.1. SDLC & V Model
  - 3.2. Testing phases
  - 3.3. Types of system testing
  - 3.4. Acceptance testing- Alpha testing, Beta testing
  - 3.5. Exploratory testing
  - 3.6. Test strategy



Copyright © Capgemini 2015. All Rights Reserved 10

## Table of Contents

- Lesson 4: Test Case Execution / Testing Metrics
  - 4.1. Introduction to Metrics – Need and definition of Metrics
  - 4.2. Metrics for Testing
  - 4.3. Types of Metrics
    - 4.3.1. Project Metrics
    - 4.3.2. Process Metrics
    - 4.3.3. Productivity Metrics
    - 4.3.4. Closure Metrics
  - 4.4. Best Practices for test case maintenance



Copyright © Capgemini 2015. All Rights Reserved 11

## References

- Student material:
  - Class Book (presentation slides with notes)
  - Lab book
- Book:
  - Testing Computer Software – Cem Kaner
  - Software Testing in the Real World – Edward Kit
  - Effective methods for Software testing – William E. Perry
  - Software Engineering -A Practitioner's Approach – Roger S. Pressman
  - Software Testing Techniques – Boris Beizer
- Web-site:
  - <http://www.softwaretesting.org>
  - <http://www.onestoptesting.com/introduction/>



Copyright © Capgemini 2015. All Rights Reserved 12

## Next Step Courses

- None



Copyright © Capgemini 2015. All Rights Reserved 13

# **Testing Concepts for V&V Automation Testing**

Lesson 1: Introduction to  
Software Testing

## Lesson Objectives

- To understand the following topics:
  - Testing Overview
  - What is Testing
  - Why testing
  - How testing
  - Testing principles
  - Objective of testing
  - Exhaustive testing
  - Limitations of software Testing
  - Psychology of testing
  - Test case
  - Test case formats
  - Other related terminologies



## 1.1: Testing Introduction

- Testing is the process of executing a program with the intent of finding errors as early as possible in SDLC
- 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.

1.1: Testing

## Need for Testing

- Contribute to the delivery of higher quality software product
- Undetected errors are costly to detect at a later stage
- Satisfied users and to lower maintenance cost



Copyright © Capgemini 2015. All Rights Reserved 4

### Why is testing becoming such a crucial activity?

1. Because applications are becoming very complex with n-tiers in an application.
2. When one tests a program one adds value to it through improved quality and reliability.
3. If not tested it can cause an unpleasant navigational error in case of a browsing applications or death or injury in case of safety critical applications.
4. End customers are becoming more demanding & conscious about quality

### Why are company's outsourcing the testing phase?

It is being realized that testing is an extremely important phase, customers today are conscious of quality as they need to be more competitive in the market.

It is being realized that the best people to test an application are the ones who have not developed the application. The testers would have the approach of a user and have an unbiased mind.

1.1: Testing

## How Testing is conducted

- By examining the users' requirements
- By reviewing the artifacts like design documents
- By examining the design objectives
- By examining the functionality
- By examining the internal structures and design
- By executing code



Copyright © Capgemini 2015. All Rights Reserved 5

### How testing is conducted?

Testing is conducted with the help of users requirements, design documents, functionality, internal structures & design, by executing code.

## 1.1: Testing Principles

- 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 - Testing must end when the economic returns cease to make it worth while i.e. the costs of testing process significantly outweigh the returns



Copyright © Capgemini 2015. All Rights Reserved 6

Although testing is itself an expensive activity, the cost of not testing is potentially much higher. The most damaging errors are those which are not discovered during the testing process and therefore remain when the system goes live.

**Limiting** - It is infeasible to test exhaustively all but the most simple or the most vital of s/w.

Here we discuss how exhaustive testing is impossible to achieve.

**Consider the following example:**

Exhaustive input testing (Black box)

Lets assume that we have written a function say  $ax^2 + bx + c = 0$

Assume 16 bit numbers

So each input is  $2^{16}$

And so total test cases is  $2^{16} \times 2^{16} \times 2^{16} = 2^{48}$  test cases which is impractical.

1.1: Testing Objective

## Objectives of Testing

- To find greatest possible number of errors with manageable amount of efforts applied over a realistic time span with a finite number of test cases



Cost effective



Copyright © Capgemini 2015. All Rights Reserved 7

We have found that we do not hesitate to change and improve well-tested code, whereas we fear changing untested code.

1.1: Testing

## Exhaustive Testing

### ▪ Exhaustive Testing

- Testing every possible input over every possible output
- Can use every possible input condition as a test case
- Is Exhaustive Testing feasible?
  - E.g. Online railway reservation system
  - Impossible to create test cases to represent all valid and invalid cases of source and destination cities



Copyright © Capgemini 2015. All Rights Reserved 8

### Exhaustive Path testing ( white box)

Even if all the paths are covered we can still not say that the program is correct for the reasons that the program may have missed some of the requirements the program may have some missed paths.

Since both exhaustive input and exhaustive path testing are not useful strategies we need to combine the elements of both to arrive at a reasonable testing strategy which we will be discussing in the next chapter.

For E.g. COBOL Compiler

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

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

1.1: Testing

## Exhaustive Testing (Contd.)

- Exhaustive testing is hence impossible
- Implications are one cannot test a program completely to guarantee that it is error free
- Objective is to therefore find maximum errors with a finite number of test cases



Copyright © Capgemini 2015. All Rights Reserved 9

1.1: Testing

## Limitations of Software Testing

- Even if we could generate the input, run the tests, and evaluate the output, we would not detect all faults
- Correctness is not checked
  - The programmer may have misinterpreted the specs, the specs may have misinterpreted the requirements
- There is no way to find missing paths due to coding errors



Copyright © Capgemini 2015. All Rights Reserved 10

1.1: Testing

## Psychology of Testing

- Test Engineers pursue defects not people
- Don't assume that no error(s) will be found
- Test for Valid and Expected as well as Invalid and Unexpected
- The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section
- Testing is extremely creative and intellectually challenging



Copyright © Capgemini 2015. All Rights Reserved 11

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 as far as a test engineer is concerned.

**Read the following example:**

A doctor asks a patient to get all the laboratory tests and if the test could not locate any problem then it is not a successful test as it did not find out the cause of the patient's problems and only wasted the laboratory fees and efforts in getting the tests done.

The psychology of people towards treating testing as the process of demonstrating that errors are not present and treating testing as the process of uncovering errors needs be discussed here

Retesting is required when errors are found. So the test cases need to be documented and retained. This gives us indication on where the errors are found most and make more investments and investigation on those areas.

Writing automated tests is harder than writing the code itself, in many cases. The most expert programmers are the best testers. When faced with seemingly mundane coding tasks, coming up with creative tests provides an intellectual challenge that expert programmers thrive on.

Beginners typically need expert assistance when writing tests. This is where pair-programming helps, because experts work side-by-side with beginners as they learn the art of testing.

**Just like coding is an art, testing is an art.**

In many organizations, testing is relegated to the least-experienced programmers. We often encounter the misconception that testing consists of people completing written checklists as they manually execute the application. This approach is completely unscalable, because it takes longer and longer for humans (monkeys?) to test every feature as the application grows.

Modern OO languages like Java are complex, particularly when it comes to dependencies between classes. One change can easily introduce bugs in seemingly unrelated classes. Gone are the days when each character-based screen is a standalone program. OO apps are far more complex and demand automated tests.

## 1.1: Testing 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”- Definition by IEEE
- In other words, a planned sequence of actions (with the objective of finding errors)



Copyright © Capgemini 2015. All Rights Reserved 13

A Test case is a planned sequence of actions.

1.1: Testing

## A good Test Case

- Has a high probability of detecting error(s)
- Test cases help us discover information
- Maximize bug count
- Help managers make ship / no-ship decisions
- Minimize technical support costs



Copyright © Capgemini 2015. All Rights Reserved 14

### Information Objectives:

So what are we trying to learn or achieve when we run tests? Here are some examples:

**Find defects.** 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.

**Help managers make ship / no-ship decisions.** Managers are typically concerned with risk in the field. They want to know about coverage (maybe not the simplistic code coverage statistics, but some indicators of how much of the product has been addressed and how much is left), and how important the known problems are. Problems that appear significant on paper but will not lead to customer dissatisfaction are probably not relevant to the ship decision.

**Minimize technical support costs.** Working in conjunction with a technical support or help desk group, the test team identifies the issues that lead to calls for support. These are often peripherally related to the product under test--for example, getting the product to work with a specific printer or to import data successfully from a third party database might prevent more calls than a low-frequency, data-corrupting crash.

**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.

**Conform to regulations.** If a regulation specifies a certain type of coverage (such as, at least one test for every claim made about the product), the test group creates the appropriate tests. If the regulation specifies a style for the specifications or other documentation, the test group probably checks the style. In general, the test group is focusing on anything covered by regulation and (in the context of *this objective*) nothing that is not covered by regulation.

**Minimize safety-related lawsuit risk.** Any error that could lead to an accident or injury is of primary interest. Errors that lead to loss of time or data or corrupt data, but that don't carry a risk of injury or damage to physical things are out of scope.

**Find safe scenarios for use of the product (find ways to get it to work, in spite of the bugs).** Sometimes, all that you're looking for is one way to do a task that will consistently work--one set of instructions that someone else can follow that will reliably deliver the benefit they are supposed to lead to. In this case, the tester is not looking for bugs. He is trying out, empirically refining and documenting, a way to do a task.

**Assess quality.** This is a tricky objective because quality is multi-dimensional. The nature of quality depends on the nature of the product. For example, a computer game that is rock solid but not entertaining is a lousy game. To assess quality -- to measure and report back on the level of quality -- you probably need a clear definition of the most important quality criteria for this product, and then you need a theory that relates test results to the definition.

For example, reliability is not just about the number of bugs in the product. It is (or is often defined as being) about the number of reliability-related failures that can be expected in a period of time or a period of use. (Reliability-related? In measuring reliability, an organization might not care, for example, about misspellings in error messages.) To make this prediction, you need a mathematically and empirically sound model that links test results to reliability. Testing involves gathering the data needed by the model. This might involve extensive work in areas of the product believed to be stable as well as some work in weaker areas. Imagine a reliability model based on counting bugs found (perhaps weighted by some type of severity) per N lines of code or per K hours of testing. Finding the bugs is important. Eliminating duplicates is important. Troubleshooting to make the bug report easier to understand and more likely to fix is (in the context of assessment) out of scope.

1.1: Testing

## A good Test Case (Contd.)

- Assess conformance to specification
- Verify correctness of the product
- Minimize safety-related lawsuit risk
- Find safe scenarios for use of the product
- Assess quality



Copyright © Capgemini 2015. All Rights Reserved 16

**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.

1.2: Test case sample format

## Test Case Proc - Format A

|  |           |                |      |
|--|-----------|----------------|------|
| Project  | Procedure |                | Date |
| Reviewer                                       |           |                | Date |
| Module / Program                               | Stage \$  | Test Case no : |      |
| <b>Process Of Test / Test Condition</b>        |           |                |      |
| Validate avg field with invalid lower boundary |           |                |      |
| <b>Test Procedure</b>                          |           |                |      |
| Calculate Total of 3 marks & average of it     |           |                |      |
| <b>Input / Test Data</b>                       |           |                |      |
| avg = 0  |           |                |      |
| <b>Expected Result</b>                         |           |                |      |
| Invalid lower boundary.                        |           |                |      |
| <b>Actual Result</b>                           |           |                |      |
| Iteration 1                                    | Tested By | Date           |      |
| Iteration 2                                    | Tested By | Date           |      |
| Iteration 3                                    | Tested By | Date           |      |
| <b>Remark (if any )</b>                        |           |                |      |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

Format A has different sections like Test condition, Test procedure, Input/Test data, Expected result, Actual result and Remarks.

Format A specifies the sample test case for calculating total and average from it. The above example demonstrates the test case for invalid lower boundary value.

1.3: Test case example

## Test Case Proc - Format B

| Test Case ID | Preconditions (If any)           | Test Condition / Scenario                       | Test Steps    | Input/Test Data | Expected Result |
|--------------|----------------------------------|---|---------------|-----------------|-----------------|
| TC_1         | Total of 3 marks & average of it | Validate avg field with invalid lower boundary  | Enter average | avg = 0         | Error1          |
| TC_2         | Total of 3 marks & average of it | Validate day field with invalid higher boundary | Enter average | avg = 101       | Error2          |
| TC_3         | Total of 3 marks & average of it | Validate day field with valid boundary          | Enter average | avg = 50        | Message1        |
| TC_4         | Total of 3 marks & average of it | Validate day field with valid boundary          | Enter average | avg = 69        | Message2        |
| TC_5         | Total of 3 marks & average of it | Validate day field with valid boundary          | Enter average | avg = 85        | Message3        |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Above example demonstrates about writing test cases for Software Testing course.

For finding grade by calculating average marks of the 3 subjects & calculating average of it.

If the average marks fall in between 40 – 59 - grade 2<sup>nd</sup>. If the average marks fall in between 60 – 74 - grade 1<sup>st</sup>. If the average marks are more than 75, grade is distinction. Detailing of the error messages & other messages should be logged in other excel sheet.

Similarly, for negative values, the expected result should generate an error message.

For all the valid cases message has to be appeared.

1.3: Test case example

## Test Case Proc - Format B

- Popup Table with error messages

| Popup Table |                          |
|-------------|--------------------------|
| Error1      | Please enter valid value |
| Error2      | Please enter valid value |
| Message1    | Grade is second class    |
| Message2    | Grade is first class     |
| Message3    | Grade is distinction     |



Copyright © Capgemini 2015. All Rights Reserved 19

Above is the popup table that lists all the error messages & other messages, correspond to the test cases that are listed in the previous slide. Detail description about the errors & messages is to be covered in popup table

For E.g. If the user is entering the invalid value, the hyperlink will prompt it to the popup table on the different sheet of the same excel document.

It will prompt a proper messages for a given test case.

1.3: Test case example

## Test Case Proc - Format B

|                              |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
|------------------------------|-----------|---------------|--------------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|--|--|
| Project Name :               |           | Prepared By : |                    |         |         |         |         |         |         |         |         |         |  |  |
| Project ID:                  |           | Prepared On   |                    |         |         |         |         |         |         |         |         |         |  |  |
| Module Name:                 |           | Reviewed By : |                    |         |         |         |         |         |         |         |         |         |  |  |
| Test Suite ID :              |           | Reviewed On : |                    |         |         |         |         |         |         |         |         |         |  |  |
| Reference TO F5              |           | Title :       |                    |         |         |         |         |         |         |         |         |         |  |  |
| Program Checklist            |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
| Conditions / Expected Result |           |               | Inspection Item ID |         |         |         |         |         |         |         |         |         |  |  |
| Conditions                   |           |               | T-01001            | T-01002 | T-01003 | T-01004 | T-01005 | T-01006 | T-01007 | T-01008 | T-01009 | T-01010 |  |  |
|                              |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
| Results                      |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
|                              |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
| Code Walkthrough             |           |               |                    |         |         |         |         |         |         |         |         |         |  |  |
| TC1                          | Tested By |               | On                 |         |         |         |         |         |         |         |         |         |  |  |
| TC2                          | Tested By |               | On                 |         |         |         |         |         |         |         |         |         |  |  |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

Format C shows a matrix that focuses on conditions, results, bug number reference across inspection item that is the test case.

In spite of different formats the test case creation highlights on 3 items: main test condition, input test data and the expected result from the test case.

1.4: Testing

## Other Terminologies

- Test Suite – A set of individual test cases/scenarios that are executed as a package, in a particular sequence and to test a particular aspect
  - E.g. Test Suite for a GUI or Test Suite for functionality
- Test Cycle – A test cycle consists of a series of test suites which comprises a complete execution set from the initial setup to the test environment through reporting and clean up.
  - E.g. Integration test cycle / regression test cycle



Copyright © Capgemini 2015. All Rights Reserved 21

**Test case** is a triplet [I, S, O] where

- I is input data
- S is state of system at which data will be input
- O is the **expected** output

**Test Suite** – Test suite is set of all test cases. Suites are usually related by the area of the application they exercise or by their priority or content.

For E.g. When you Login to the screen, some functionalities like validating user name, password with different invalid inputs can act as Test suites.

E.g. In case of ATM machine, deposit, withdraw, balance check are separate test suites that carry out different test cases

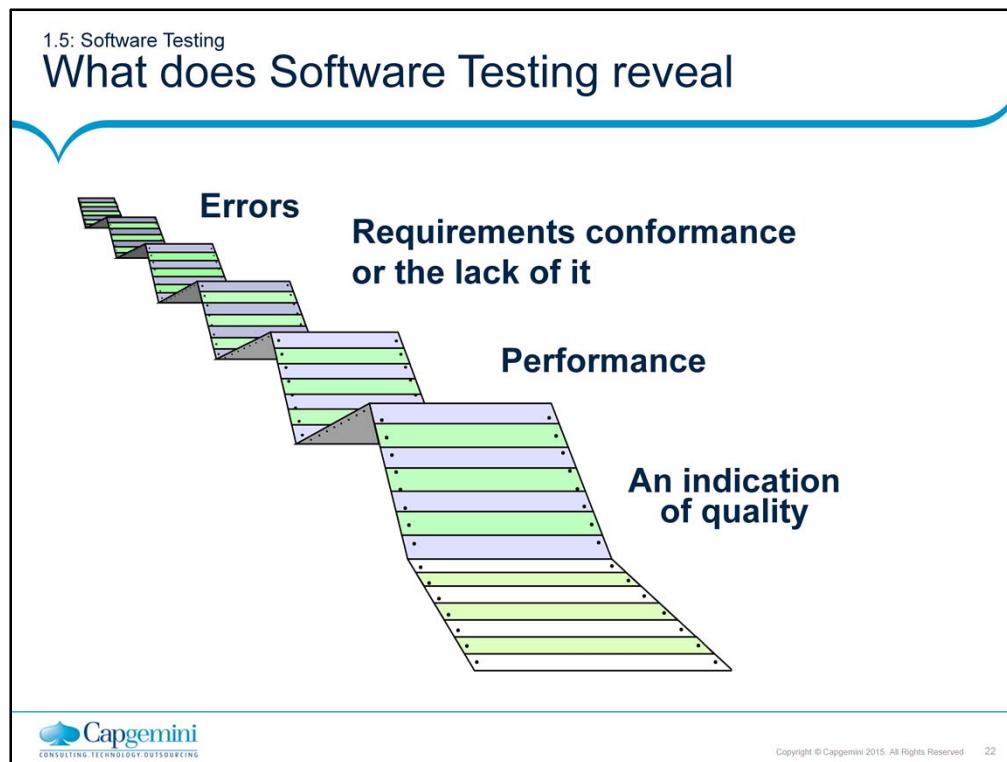
**Test Cycle** – It's a combination of series of test suites.

For E.g. The Test Cycle for the same application is combination of multiple Test Suites like, Functional validations, Database validations, GUI validations, etc. form the Test Cycle.

E.g. Combination of all test suites like deposit, withdraw, balance check in case of ATM machine, forms a complete cycle

**Test Scenario** — A set of test cases that ensure that the business process flows are tested from end to end. They may be independent tests or a series of tests that follow each other, each dependent on the output of the previous one. The terms "test scenario" and "test case" are often used synonymously.

Test cases are not randomly selected. Instead even they need to be designed.a



Software testing exposes all the possible, noticed & unnoticed Errors.

It also takes care of the requirements conformance with respect to the end product.

It checks for the performance test.

The very crucial aim of all of these activities is Quality. It ensures the product in terms of quality.

## Summary

- In this lesson, you have learnt:
  - Testing is an extremely creative & intellectually challenging task
  - No software exists without bug
  - Testing is conducted with the help of users requirements, design documents, functionality, internal structures & design, by executing code
  - The cost of not testing is potentially much higher
  - Testing is in a way a destructive process
  - A successful test case is one that brings out an error in program



## Review Question

- Question 1: \_\_\_\_\_ is a planned sequence of actions
  
- Question 2: One cannot test a program completely to guarantee that it is error free
  - Option: True / False
  
- Question 3: One can find missing paths due to coding errors
  - Option: True / False



## Review Question: Match the Following

Economics of limiting

2. Testing

3. A good test case

4. Use every possible  
input condition as a  
test case

A. Driving

B. Exhaustive testing

C. Limiting

D. Test cycle

E. Comparing outputs with  
specified or intended

F. Maximize bug count



## **Testing Concepts for V&V Automation Testing**

Lesson 2: Types of Testing Techniques & Test Case Design

## Lesson Objectives

- To understand the following topics:
  - Types of testing techniques
  - Static testing - Self Review, Code inspection, Walkthrough Desk check.
  - Dynamic testing - White Box, Black Box.
  - Test Case Construction
  - Test Data Preparation
  - Test Case Design Techniques – Black Box Testing
  - What is positive testing and negative testing
  - Example of positive and negative testing
  - Scenarios of positive and negative testing
  - What is basic and alternate testing
  - Importance of writing positive,negative,basic,alternate tests while designing test cases



2.1: Testing Techniques

## Types

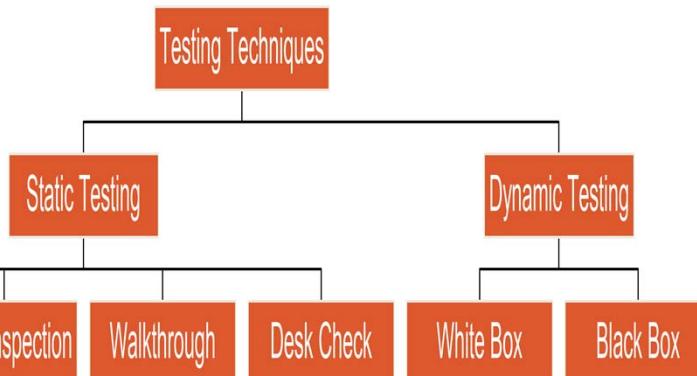
- Static Testing - Testing a software without execution on a computer. Involves just examination/review and evaluation
- Dynamic Testing - Testing software through executing it



Copyright © Capgemini 2015. All Rights Reserved 3

2.1: Testing Techniques  
**Types (Contd.)**

Types of Testing Techniques



2.1: Static Testing

## Introduction

- Static Testing is a process of reviewing the work product and reviewing is done using a checklist
- Static Testing helps weed out many errors/bugs at an early stage
- Static Testing lays strict emphasis on conforming to specifications
- Static Testing can discover dead codes, infinite loops, uninitialized and unused variables, standard violations and is effective in finding 30-70% of errors



Copyright © Capgemini 2015. All Rights Reserved 5

2.1: Static Testing

## Introduction (Contd.)

- Static Testing Methods
  - Self Review
  - Code Inspection
  - Walk Through
  - Desk Checking



Copyright © Capgemini 2015. All Rights Reserved 6

## 2.1: Static Testing Self Review

- Self review is done by the person who is responsible for a particular program code
- It is more of reviewing the code in informal way
- It is more like who writes the code, understands it better
- Self review is to be done by the programmer when he builds a new code
- There are review checklists that helps programmer to verify with the common errors regarding the program code



Copyright © Capgemini 2015. All Rights Reserved 7

### ERROR CHECKLIST FOR INSPECTION

#### 1. Data Reference Errors

For each array reference, is each subscript value within defined bounds?

**Dangling reference problem:** arises when the lifetime of a pointer is greater than the lifetime of a referenced storage.

If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

#### 2. Data Declaration errors

Is each variable has been assigned correct length, type, and storage class?

Are there any variables with similar names (not an error but may have been confused with in the program)?

#### 3. Computation errors

Are there any computations using same data types but different lengths?

Is an underflow or overflow occurs during computation?

Division by zero and square root of a negative number errors.

Are the order of evaluation and precedence of operators correct?

#### 4. Comparison errors:

Are there any mixed mode computations or comparisons between variables of different lengths?

Is the comparison operator correct? Does each Boolean expression state what it is supposed to do? Are the operands of a Boolean operator Boolean?

2.1: Static Testing

## Code Review Checklist

- Data Reference Errors
  - Is a variable referenced whose value is unset or uninitialized?
- Data Declaration Errors
  - Have all variables been explicitly declared?
  - Are variables properly initialized in declaration sections?
- Computation errors
  - Are there any computations using variables having inconsistent data types?
  - Is there any mixed mode computations?
- Comparison errors
  - Are there any comparisons between variables having inconsistent data types?

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

### ERROR CHECKLIST FOR INSPECTION

#### 1. Data Reference Errors

For each array reference, is each subscript value within defined bounds?

**Dangling reference problem:** arises when the lifetime of a pointer is greater than the lifetime of a referenced storage.

If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

#### 2. Data Declaration errors

Is each variable has been assigned correct length, type, and storage class?

Are there any variables with similar names (not an error but may have been confused with in the program)?

#### 3. Computation errors

Are there any computations using same data types but different lengths?

Is an underflow or overflow occurs during computation?

Division by zero and square root of a negative number errors.

Are the order of evaluation and precedence of operators correct?

#### 4. Comparison errors:

Are there any mixed mode computations or comparisons between variables of different lengths?

Is the comparison operator correct? Does each Boolean expression state what it is supposed to do? Are the operands of a Boolean operator Boolean?

2.1: Static Testing

## Code Review Checklist

- Control Flow errors
  - Will every loop eventually terminate?
  - Is it possible that, because of condition upon entry, a loop will never execute?
- Interface errors
  - Does the number of parameters received by these module equals the number of arguments sent by calling modules?
  - Also is the order correct?
- Input/output errors
  - All I/O conditions handled correctly?

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

### ERROR CHECKLIST FOR INSPECTION

#### **5. Control Flow errors:**

Will the program, module or subroutine eventually terminate?

Is there any nonexhaustive decision? Correct and appropriate number of END statements for group statements.

#### **6. Interface errors:**

Do the attributes (Size, type and unit system if any) of parameters match with that of the arguments?

Does a subroutine alter a parameter that is intended to be only an input value?

#### **7. Input/output errors:**

Are all the files opened properly with correct attribute in the open statement?

Are end-of-file conditions detected and handled properly? Does the format specification agree with the information in the I/O statement?

2.1: Static Testing

## Code Inspection

- 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.
- The objective is to find errors but not solutions to the errors
- An inspection team usually consists of:
  - A moderator
  - A programmer
  - The program designer
  - A test specialist



Copyright © Capgemini 2015. All Rights Reserved 10

### **Moderator duties includes:**

1. Distribution materials,
2. Scheduling the Inspection Session
3. Leading the session
4. Recording all errors found
5. Ensures that the errors found are subsequently corrected.

2.1: Static Testing

## Code Inspection

- The moderator distributes the program's listing and design specification to the group well in advance of the inspection session
- During the inspection
  - The programmer narrates the logic of the program, statement by statement
  - During the discourse, questions are raised and pursued to determine if errors exist
  - The program is analyzed with respect to a check list of historically common programming errors



Copyright © Capgemini 2015. All Rights Reserved 11

Code inspection focuses on discovering errors and not correcting them. The Inspection process is the way of identifying error prone sections early, helping to concentrate on the most sensitive sections during testing process.

2.1: Static Testing

## Code Inspection

- Code Inspection Helps in
  - Detect Defects
  - Conformance to standards/spec
  - Requirements Transformation into product



Copyright © Capgemini 2015. All Rights Reserved 12

2.1: Static Testing

## Code Walkthrough

- Code Walkthrough is a set of procedures and error detection techniques for group reading.
- Like code inspection it is also a 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.
- Solutions are also suggested by team members.



Copyright © Capgemini 2015. All Rights Reserved 13

### Walkthrough meeting.

#### This meeting will include following members:

1. A highly experienced programmer
2. A programming language expert
3. A new programmer
4. The person who maintains the program
5. Some person from a different project
6. Some one from the same team as a programmer.

#### Walk through procedure

1. The designer simulates the program.
2. She/he shows, step by step what the program will do with the test data supplied by the reviewers.
3. The simulation shows how different pieces of the system interact and can expose awkwardness, redundancy and many missed details.
4. Different from inspection that it needs the participants to be ready with test cases.

2.1: Static Testing

## Code Walkthrough

- Walkthrough helps in
  - Approach to Solution
  - Find omission of requirements
  - Style / Concepts Issues
  - Detect Defects
  - Educate Team Members



Copyright © Capgemini 2015. All Rights Reserved 14

Tester prepares test cases. During meeting, each test case is manually executed. Test data are walked through the logic of the program.

2.1: Static Testing

## Desk Checking

- Human error detection technique
- Viewed as a one person inspection or walkthrough
- A person reads a program and checks it with respect to an error list and/or walks test data through it
- Less effective technique
- Best performed by the person other than the author of the program



Copyright © Capgemini 2015. All Rights Reserved 15

It is the dry run of the program. It is completely informal process. Desk checking is best performed by a person other than the author of the program.

E.g. Two programmers might swap the program rather than desk checking their own programs.

It is less effective than inspection and walkthrough of the code.

2.1: Dynamic Testing

## Comparison

- White Box Test Techniques
  - Code Coverage
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Loop Testing
  - Code complexity
  - Cyclomatic Complexity
  - Memory Leakage

- Black Box Test Techniques
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Use Case / UML
  - Error Guessing
  - Cause-Effect Graphing
  - State Transition Testing



Copyright © Capgemini 2015. All Rights Reserved 16

2.1: Dynamic Testing

## White Box Test Techniques

- White box is logic driven testing and permits Test Engineer to examine the internal structure of the program
- Examine paths in the implementation
- Make sure that each statement, decision branch, or path is tested with at least one test case
- Desirable to use tools to analyze and track Coverage
- White box testing is also known as structural, glass-box and clear-box



Copyright © Capgemini 2015. All Rights Reserved 17

2.1: Dynamic Testing

## White Box Test Techniques

- White Box Test Techniques
  - Code Coverage
    - Statement Coverage
    - Decision Coverage
    - Condition Coverage
    - Loop Testing
  - Code complexity
  - Memory Leakage

**White box: structure/path**

```

graph TD
    If((If)) -- false --> Else((Else))
    If -- true --> Then((Then))
    Else --> End1((End))
    Then --> Case((Case))
    Case -- 1 --> A((A))
    Case -- 2 --> B((B))
    Case -- 3 --> C((C))
    Case -- 4 --> D((D))
    Case -- 5 --> E((E))
    A --> End2((End))
    B --> End2
    C --> End2
    D --> End2
    E --> End2
  
```

Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Using white box testing methods, you can derive test cases that

- (1) guarantee that all independent paths within a module have been excised at least once
- (2) exercise all logical decisions on their true and false sides
- (3) execute all loops at their boundaries and within their operational bounds
- (4) exercise internal data structures to ensure their validity.

2.1:

Dynamic Testing

## Code Coverage

- Measure the degree to which the test cases exercise or cover the logic (source code) of the program
- Types
  - Statement Coverage
  - Decision Coverage
  - Conditional Coverage
  - Loop Testing



Copyright © Capgemini 2015. All Rights Reserved 19

Code coverage is also known as logic coverage. The goal is to execute every statement of the code at least once. Test engineers can derive test cases using

White box testing methods, that

1. All independent paths within a module are traversed at least once
2. Exercise all logical decisions on their true and false sides
3. Execute all loops at their boundaries and within operational bounds
4. Exercise internal data structures to ensure their validity.

2.1: Dynamic Testing

## Statement Coverage

- Test cases must be such that all statements in the program is traversed at least once
- Consider the following snippet of code

```
void procedure(int a, int b, int x)
```

```
{    If (a>1) && (b==0)
        { x=x/a; }
    If (a==2 || x>1)
        { x=x+1; }
}
```



CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

The goal is to execute every statement in the program at least once. Every statement must be executed.

2.1: Dynamic Testing

## Statement Coverage

- Test Case:  $a=2, b=0, x=3$ .
- Every statement will be executed once.
- But only path ACE will be covered and path ABD,ACD,ABE will not be covered.

```

graph TD
    A((a > 1  
AND  
b = 0)) -- Yes --> C[x = x/a]
    C --> B((a = 2  
Or  
x > 1))
    B -- Yes (a = 2) --> E[x = x + 1]
    E --> D[x = x + 1]
    B -- No (x > 1) --> B
    B -- No (a > 1 AND b = 0) --> A

```

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Every statement can be executed by writing a single test case. This case covers only ACE path.

This criteria is weak one. Since it is not considering other paths to traverse. So the path ABD, ACD, ABE would go undetected.

2.1: Dynamic Testing

## Statement Coverage

- In the above code one test case is sufficient to execute each of the two if statements at least once:
  - Test Case : a=2 , b=0 , x=3
  - (Decision1 is True, Decision2 is True)
- However this test case does not help in detecting many of the many of the bugs which may go unnoticed as the false outcomes of the conditions  $a>1$  &  $b=0$  ,  $a=2$  or  $x>1$  are not tested



Copyright © Capgemini 2015. All Rights Reserved 22

2.1: Dynamic Testing

## Decision Coverage

- Test Case 1:  $a=2$ ,  $b=0$ ,  $x>1$
- (Decision1 is True, Decision2 is True) (Path ACE)
- Test Case 2:  $a<=1$ ,  $b!=0$ ,  $x<=1$
- (Decision1 is False, Decision2 is False) (Path ABD)

Copyright © Capgemini 2015. All Rights Reserved 23

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Decision coverage can cover two test cases covering paths ACE and ABD. Even if the above test cases satisfy decision coverage it still does not cover the path ACD and path ABE. Hence decision coverage though stronger criteria than statement it is still weak. There is only 50 percent chance that we would explore the path.

2.1: Dynamic Testing

## Condition Coverage

- Test cases are written such that each condition in a decision takes on all possible outcomes at least once.

Test Case1 : a=2, b=0, x=3  
(Condition1 is True, Condn2 is True)  
(Path ACE)

Test Case2: a=3, b=0, x=0  
(Condn1 is True, Condn2 is False, Condn3 is False)  
(Path ACD)

```

graph TD
    A((a > 1  
AND  
b = 0)) -- Yes --> C[x = x/a]
    A -- No --> B((a = 2  
Or  
x > 1))
    B -- Yes --> E[x = x + 1]
    B -- No --> D(( ))
  
```

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

Condition testing is a test case design method that exercises the logical conditions contained in a program module. A simple condition is a Boolean variable or a relational expression. Relational operator is one of the following <, <=, =, not =, > , =>.

A compound condition is composed of two or more simple conditions, Boolean operators, and parentheses.

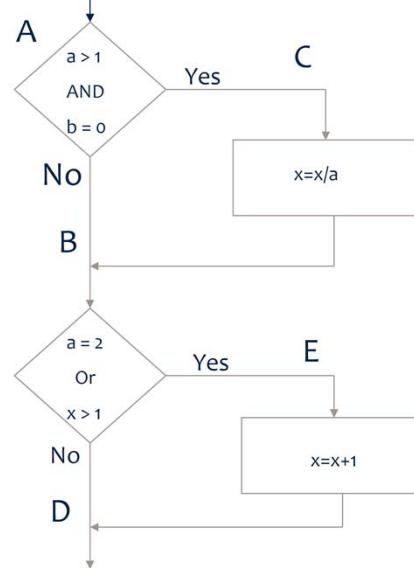
Condition coverage focuses on testing each condition in a program. The purpose of the condition testing is to detect not only errors in the conditions of a program but also other errors in the program.

2.1: Dynamic Testing

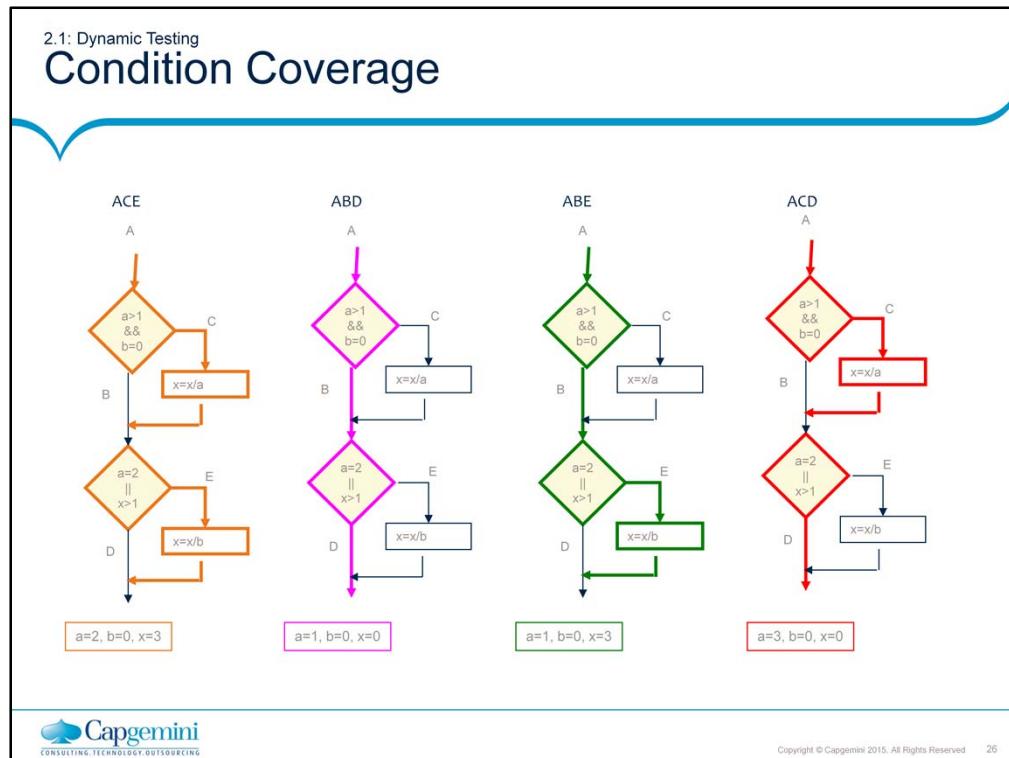
## Condition Coverage

- Test Case3 :  $a=1$ ,  $b=0$ ,  $x=3$
- (Condition1 is False, Condition2 is True)
- (Path ABE)

- Test Case4:  $a=1$ ,  $b=1$ ,  $x=1$
- (Condition1 is False, Condition2 is False)
- (Path ABD)



Copyright © Capgemini 2015. All Rights Reserved 25



### What does “coverage” mean?

- NOT all possible combinations of data values or paths can be tested
- Coverage is a way of defining how many of the paths were actually exercised by the tests
- Coverage goals can vary by risk, trust, and level of test

In the above diagrams, each condition in decision takes all possible outcomes at least once.

2.1: Dynamic Testing

## Loop Testing

- Loops testing is a white box testing technique that focuses exclusively on validity of Loop construct
- Types of loops
  - Simple Loop
  - Nested Loop
  - Concatenated Loop
  - Spaghetti Loop



Copyright © Capgemini 2015. All Rights Reserved 27

Loops are cornerstones for many algorithms. It is very important to test every loop carefully.

2.1: Dynamic Testing

## Loop Testing

- Simple Loop Testing Procedure:
  - skip the entire loop
  - only one pass through the loop
  - make 2 passes through loop
  - $m$  passes through loop where  $m < n$
  - $n-1, n, n+1$  passes through the loop
- Where  $n$  is the maximum number of allowable passes through the loop

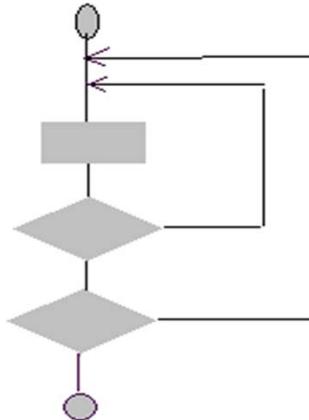
The first flowchart shows a decision diamond pointing to a process rectangle, which then loops back to the decision diamond. The second flowchart shows a process rectangle followed by a decision diamond, with a feedback loop from the decision diamond back to the process rectangle.

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 28

## 2.1: Dynamic Testing Loop Testing

- Nested Loop Testing Procedure:
  - start at the innermost loop
  - conduct simple loop test for the innermost loop
  - work outward, conducting tests for the next loop but keeping all other loops at minimum
  - continue until all the outer loops are tested



Copyright © Capgemini 2015. All Rights Reserved 29

Number of test cases increases as the loop gets extended.

2.1: Dynamic Testing

## Loop Testing

- Concatenated Loop Testing Procedure:
  - If each loop is independent of the other, test them as simple loops, else test them as nested loops

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

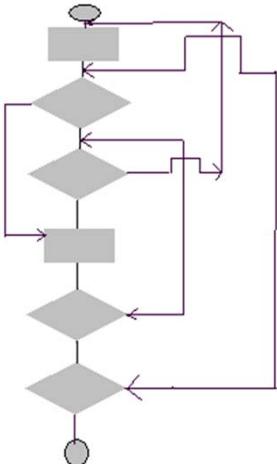
Copyright © Capgemini 2015. All Rights Reserved 30

If the loops are independent of others, concatenated loops can be tested using simple loop testing approach. However, if the loops are concatenated and are not independent, the loop counter for loop 1 is used as the initial value for loop 2.

2.1: Dynamic Testing

## Loop Testing

- Spaghetti loops Testing Procedure:
  - Redesign using structured constructs



**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 31

These are the unstructured loops. These loops should be redesigned to structured constructs.

2.1: Dynamic Testing

## Basis Path Testing

- Basis Path Testing is a white box testing technique that enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining the basis set of execution paths
- Test Cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing



Copyright © Capgemini 2015. All Rights Reserved 32

2.1: Dynamic Testing

## Flow Graph

- Main tool for test case identification
- Shows the relationship between program segments , which is the sequence of statements having the property that if the first member of the sequence is executed then all other statements in that sequence will also be executed



Copyright © Capgemini 2015. All Rights Reserved 33

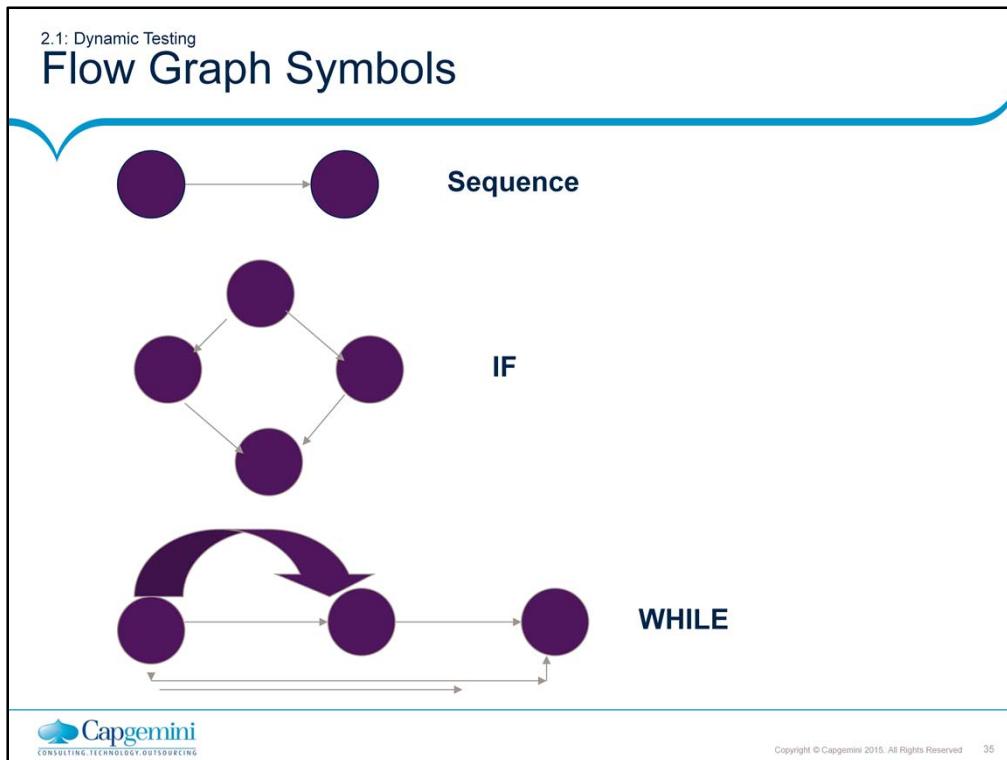
2.1: Dynamic Testing

## Flow Graph Symbols

- Nodes represent one program segment
- Areas bounded by edges and nodes are called regions
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition
- Each node containing a condition called a predicate node



Copyright © Capgemini 2015. All Rights Reserved 34



Sequence Flow – indicates the statements to be executed in a defined sequence

If – Sequence of statements will be executed depending on the condition match

While - Sequence of statements will be executed depending on the condition match, another way of achieving If flow.

Flow graph is the notation for the representation of the control flow. It depicts logical control flow with the help of notations.

Each circle is called a flow graph node, represents one or more procedural statements. The arrows on the flow graph , called edges or links, represent flow of control and are analogous to flowchart arrows. An edge must terminate at a node, even if the node does not represent any procedural statements.

Areas bounded by edges and nodes are called regions. When counting regions, we include the area out-side the graph as a region.

When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated. A compound condition occurs when one or more Boolean operators (logical OR, AND, NAND, NOR) is present in a conditional statement.

Each node containing a condition called a predicate node. It is characterized by two or more edges emanating from it.

2.1: Dynamic Testing

## Cyclomatic Complexity

- Cyclomatic Complexity is a software metric that provides a quantitative measure of logical complexity of a program
- When Used in the context of the basis path testing method, value for cyclomatic complexity defines number of independent paths in basis set of a program
- Also provides an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once
- Cyclomatic complexity is often referred to simply as program complexity, or as McCabe's complexity



Copyright © Capgemini 2015. All Rights Reserved 36

### Introduced by Thomas McCabe in 1976:

1. Count the regions of the flow graph (including the exterior)
2. Or compute by  $e-n+2$  This is called Cyclomatic Complexity
3. The number of paths to test , all decision options are tested

### How many paths (McCabe's technique for units)?

Cyclomatic complexity defines the number of independent paths. This provides minimum number of tests to be conducted to ensure all the statements have been executed at least once.

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

Cyclomatic complexity is a useful metric for predicting those modules that are likely to be error prone. Use it for test planning as well as test case design.

Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.

When used in context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us the upper bound of the number of tests that must be conducted to ensure that all statements have been executed at least once.

**2.1: Dynamic Testing**

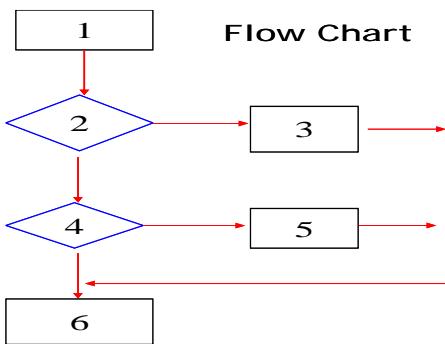
## Calculating Cyclomatic Complexity

- The cyclomatic complexity of a software module is calculated from a flow graph of the module , when used in context of the basis path testing method
- Cyclomatic Complexity  $V(G)$  is calculated one of the three ways:
  - $V(G) = E - N + 2$  , where E is the number of edges and N = the number of nodes of the graph
  - $V(G) = P+1$ , where P is the number of predicate nodes
  - $V(G) = R$  , where number of region in the graph

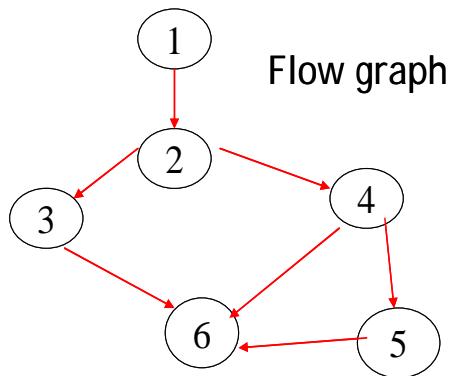
**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 37

Here, a flow chart is used to depict program control structure. Flow chart is mapped into corresponding flow graph. Each circle is called as flow graph node, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node.



A flow chart depicts program control structure.  
Flow chart is mapped into flow graph.  
A sequence of process boxes and a decision diamond are map into a single node.

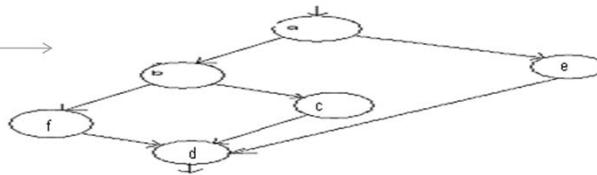


Each circle is called as graph node.  
Arrows are called as edges.  
The arrows on the flow graph is called edges, represents flow of control. An edge must be terminated at a node.

2.1: Dynamic Testing

## Calculating Cyclomatic Complexity : Example

In the given figure a and b are predicate nodes



- Cyclomatic Complexity,  $V(G)$  for a flow Graph G is  $V(G) = E - N + 2$
- E = Number of Edges in the graph (7 in the above figure)
- N = number of flow graph Nodes (6)
- R = number of Regions (3)
- Hence  $V(G) = 7-6+2 = 3$
- $V(G)$  can also be calculated as  $V(G) = P+1$ , where P is the number of predicate nodes. Here  $V(G) = 2+1 = 3$
- Also  $V(G)$  can be calculated as  $V(G) = R$  hence  $V(G) = 3$



Copyright © Capgemini 2015. All Rights Reserved 38

The value of  $V(G)$  provides the number of linearly independent paths through the program.

Here, in the above graph, value of  $V(G)$  is 3. That is, minimum 3 test cases must be designed to guarantee coverage of all program statements.

2.1: Dynamic Testing

## Calculating Cyclomatic Complexity : Example

**Flow graph**

Edges(E) = 7 Nodes(N) = 6  
Regions (R) = 3 , Predicate nodes (P) = 2  
 $CC = E - N + 2 = 7 - 6 + 2 = 3$   
 $CC = P + 1 = 2 + 1 = 3$   
 $CC = R = 3$

- Cyclomatic complexity gives minimum number of test cases to be performed to cover all the program statements.

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 39

In the above graph, the cyclomatic complexity is 3. That means, there are 3 independent paths in this program. Independent path is any path through the program that has set of statements or a condition.

Path 1: 1-2-3-6

Path 2: 1-2-4-5-6

Path 3: 1-2-4-6

In order to have complete coverage of this code, there are minimum 3 test cases required which traversed thorough the above paths.

2.1: Dynamic Testing

## Memory Leak

- Memory leak is present whenever a program loses track of memory.
- Memory leaks are most common types of defect and difficult to detect
- Performance degradation or a deadlock condition occurs
- Memory leak detection tools help to identify
  - memory allocated but not deallocated
  - uninitialized memory locations



Copyright © Capgemini 2015. All Rights Reserved 40

2.1: Dynamic Testing

## Memory Leak

- Find the error in the following snippet of code

```
void read_file(char*);  
void test(bool flag)  
{  
    char* buf = new char[100];  
    if (flag) {  
        read_file(buf);  
        delete [] buf;  
    }  
}
```



Copyright © Capgemini 2015. All Rights Reserved 41

This problem occurs if a program fails to free objects that are no longer in use. The code leaks 100 bytes of memory every time the function test is called with the argument flag as false.

If a program continues to leak memory, its performance degrades. Its runtime memory footprint continues to increase and it spends more and more time in swapping and can eventually run out of memory.

You can use the garbage collection library to fix these errors.

Same memory pointer is deleted more than once:

```
void test3()  
{  
    int* p1 = new int;  
    int* p2 = p1;  
    delete p1;  
    delete p2; // deleting again!  
}
```

2.1: Dynamic Testing

## Memory Fragmentation and Overwrites

- Memory Fragmentation

- caused by frequent allocation and deallocation of memory
- can degrade an application's performance
- occurs when a large chunk of memory is divided into much smaller, scattered pieces
- May not be never allocated again

- Memory Overwrites

- too little memory is allocated for an object
- can include memory corruption and intermittent failures.
- program may work correctly some times and fail at other times



Copyright © Capgemini 2015. All Rights Reserved 42

### Memory Fragmentation

This can be caused by frequent allocation and deallocation of memory and can degrade an application's performance. Memory fragmentation occurs when a large chunk of memory is divided into much smaller, scattered pieces. These smaller discontiguous chunks of memory may not be of much use to the program and may never be allocated again. This can result in the program consuming more memory than it actually allocates.

### Memory Overwrites

This problem occurs when too little memory is allocated for an object. Consequences can include memory corruption and intermittent failures. The program may work correctly some times and fail erratically at other times.

**Memory Overwrites:** Once a block of memory has been allocated, it is important that the program does not attempt to write any data past the end of the block or write any data just before the beginning of the block. Even writing a single byte just beyond the end of an allocation or just before the beginning of an allocation can cause disaster. It is a possible candidate for turning on overflow buffers.

2.1: Dynamic Testing

## Black Box Test Techniques

- Black box is data-driven, or input/output-driven testing
- The Test Engineer is completely unconcerned about the internal behavior and structure of program
- Black box testing is also known as behavioral, functional, opaque-box and closed-box



Copyright © Capgemini 2015. All Rights Reserved 43

Black Box At Different Levels – Unit, Subsystem and System.

A black box is just a bigger box with more input, functionality, and output.

2.1: Dynamic Testing

## Black Box Test Techniques

I can't reach  
ALL of the code!



Copyright © Capgemini 2015. All Rights Reserved 44

Trainer Notes

2.1: Dynamic Testing

## Black Box Test Techniques

- Tests are designed to answer the following questions:
  - How is functional validity tested ?
  - What classes of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - What effect will specific combinations of data have on system operations?



Copyright © Capgemini 2015. All Rights Reserved 45

Black Box testing also called behavioral testing, focuses on the functional requirements of the software. Black box testing is not an alternative to white box

techniques. Rather it is complementary approach that is likely to uncover a different class of errors than white box methods.

Black box testing attempts to find errors in the following categories

- (1) incorrect or missing functions
- (2) interface errors
- (3) errors in data structures or external database access
- (4) behavior or performance errors
- (5) initialization errors.

2.1: Dynamic Testing

## Black Box Test Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- Error Guessing
- Cause Effect Graphing
- State transition testing



Copyright © Capgemini 2015. All Rights Reserved 46

Programmers are logical thinkers, so they catch many of the “logical” defects. Real users are NOT necessarily logical.  
Real environmental circumstances are often illogical.

2.1: Dynamic Testing

## Equivalence Partitioning

- This method divides the input domain of a program into categories of data for deriving test cases
- Identify equivalence classes - the input ranges which are treated the same by the software
  - Valid classes: legal input ranges
  - Invalid classes: illegal or out of range input values
- The aim is to group and minimize the number of test cases required to cover these input conditions



Copyright © Capgemini 2015. All Rights Reserved 47

For those familiar with elementary statistical techniques, EP is very much similar to **class intervals** and **tally marks analysis**.

An ideal test case single handedly uncovers a class of errors (e.g. incorrect processing of all character data) that might require many cases to be executed before general error is observed. EP strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.

An equivalence class represents a set of valid or invalid states for input condition. An input condition is either a specific numeric value , a range of values , a set of related values or a Boolean condition.

2.1: Dynamic Testing

## Equivalence Partitioning

Assumption:

- If one value in a group works, all will work
- One from each partition is better than all from one
- Thus it consists of two steps:
  - Identify the Equivalence class
  - Write test cases for each class



Copyright © Capgemini 2015. All Rights Reserved 48

For those familiar with elementary statistical techniques, EP is very much similar to **class intervals** and **tally marks analysis**.

2.1: Dynamic Testing

## Equivalence Partitioning

Examples of types of equivalence classes

- If an input condition specifies a continuous range of values, there is one valid class and two invalid classes
- Example: The input variable is a mortgage applicant's income. The valid range is \$1000/mo. to \$75,000/mo
  - Valid class: {1000 > = income < = 75,000}
  - Invalid classes: {income < 1000}, {income > 75,000}



Copyright © Capgemini 2015. All Rights Reserved 49

If an input condition specifies a set of values, there is reason to believe that each is handled differently in the program.

e.g., Type of Vehicle must be Bus, Truck, Taxi). A valid equivalence class would be any one of the values and invalid class would be say Trailer or Van.

2.1: Dynamic Testing

## Equivalence Partitioning

- If an input condition specifies that a variable, say count, can take range of values(1 - 999)

A horizontal line with three segments. The first segment is labeled "Invalid" in red. The middle segment is labeled "Valid" in red. The third segment is labeled "Invalid" in red.

**Identify - one valid equivalence class ( $1 < \text{count} < 999$ )**  
- two invalid equivalence classes ( $\text{count} < 1$ ) & ( $\text{count} > 999$ )

A number line with tick marks at 0, 1, 999, and 1000. Blue arrows point downwards from each of these tick marks. The region between 1 and 999 is labeled "Valid". The regions before 1 and after 999 are labeled "Invalid".

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 50

**Equivalence classes may be defined according to the following guidelines.**

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, one valid and two invalid EC are defined.
3. If an input condition specifies a member of a set, one valid and one invalid EC are defined.
4. In an input condition is Boolean, one valid and one invalid class are defined.

2.1: Dynamic Testing

## Equivalence Partitioning

- If a “must be” condition is required, there is one valid equivalence class and one invalid class
- Example: The mortgage applicant must be a person
  - Valid class: {person}
  - Invalid classes:{corporation, ...anything else...}



Copyright © Capgemini 2015. All Rights Reserved 51

If we have to test function **int Max (int a , int b)** the Equivalence Classes for the arguments of the functions will be

| Arguments | Valid Values             | Invalid Values     |
|-----------|--------------------------|--------------------|
| <hr/>     |                          |                    |
| A         | -32768 <= Value <= 32767 | < - 32768 , >32767 |
| B         | -32768 <= Value <= 32767 | < - 32768 , >32767 |

2.1: Dynamic Testing

## Equivalence Partitioning

### Example

- If we have to test function int Max(int a , int b) the Equivalence Classes for the arguments of the functions will be

| Arguments | Valid Values             | Invalid Values     |
|-----------|--------------------------|--------------------|
| A         | -32768 <= Value <= 32767 | < - 32768 , >32767 |
| B         | -32768 <= Value <= 32767 | < - 32768 , >32767 |



Copyright © Capgemini 2015. All Rights Reserved 52

2.1: Dynamic Testing

## Boundary Value Analysis

- “Bugs lurk in corners and congregate at boundaries ....” *Boris Beizer*
- Boundary Conditions are those situations directly on, above, and beneath the edges of input equivalence classes and output equivalence classes.
- Boundary value analysis is a test case design technique that complements Equivalence partitioning.
- Test cases at the boundary of each input Includes the values at the boundary, just below the boundary and just above the boundary.



Copyright © Capgemini 2015. All Rights Reserved 53

BVA is not as simple as it sounds, because boundary conditions may be subtle and difficult to identify.

The method does not test combinations of input conditions.

2.1: Dynamic Testing

## Boundary Value Analysis

- From previous example, we have the valid equivalence class as ( $1 < \text{count} < 999$ )
- Now, according to boundary value analysis, we need to write test cases for  $\text{count}=0$ ,  $\text{count}=1$ ,  $\text{count}=2$ ,  $\text{count}=998$ ,  $\text{count}=999$  and  $\text{count}=1000$  respectively

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 54

Guidelines for BVA are similar in many respects to those provided for EP:

1. If an input condition specifies a range bounded by values  $a$  and  $b$ , test cases should be designed with values  $a$  and  $b$  as well as just above and just below.
2. If an input condition specifies a number of values test cases should be developed that exercise the minimum and maximum numbers. Values just above and below min and max are also tested.
3. Applying guidelines 1 and 2 output conditions. For example, assume that a temperature vs. pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and Min) allowable number of table entries.
4. If internal program data structures have prescribed boundaries (e.g.- an array has defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

2.1: Dynamic Testing

## Boundary Value Analysis

- Guidelines
  - If an input condition specifies a range of values A and B, test cases should be designed with values A and B, just above and just below A and B respectively
  - Similarly with a number of values

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 55

If we have to test function **int Max (int a , int b)** the Equivalence Classes for the arguments of the functions will be

| Arguments | Valid Values             | Invalid Values    |
|-----------|--------------------------|-------------------|
| A         | -32768 <= Value <= 32767 | < -32768 , >32767 |
| B         | -32768 <= Value <= 32767 | < -32768 , >32767 |

2.1: Dynamic Testing

## Boundary Value Analysis

- Example :

- If we have to test function int Max(int a , int b) the Boundary Values for the arguments of the functions will be

| Arguments | Valid Values                 | Invalid Values |
|-----------|------------------------------|----------------|
| A         | -32768, -32767, 32767, 32766 | -32769,32768   |
| B         | -32768, -32767, 32767, 32766 | -32769,32768   |



Copyright © Capgemini 2015. All Rights Reserved 56

2.1: Dynamic Testing

## Error Guessing

- Based on experience and intuition one may add more test cases to those derived by following other methodologies
- It is an ad hoc approach
- The basis behind this approach is in general people have the knack of “smelling out” errors



Copyright © Capgemini 2015. All Rights Reserved 57

Here mention about Myers Probability study that the probability of errors remaining in the program is proportional to the number of errors that have been found so far, which provides a rich source for productive error guessing.

2.1: Dynamic Testing

## Error Guessing

- Make a list of possible errors or error-prone situations and then develop test cases based on the list
- Defects' history are useful
- Probability that defects that have been there in the past are the kind that are going to be there in the future
- Some examples :
  - Empty or null lists/strings
  - Zero occurrences
  - Blanks or null character in strings
  - Negative numbers



Copyright © Capgemini 2015. All Rights Reserved 58

Trainer Notes

2.1: Dynamic Testing

## Error Guessing

- Example : Suppose we have to test the login screen of an application
- An experienced test engineer may immediately see if the password typed in the password field can be copied to a text field which may cause a breach in the security of the application
- Error guessing testing for sorting subroutine situations
  - The input list empty
  - The input list contains only one entry
  - All entries in the list have the same value
  - Already sorted input list



Copyright © Capgemini 2015. All Rights Reserved 59

Trainer Notes

2.1: Dynamic Testing

## Cause Effect Graphing

- A testing technique that aids in selecting, in a systematic way, a high-yield set of test cases that logically relates causes to effects to produce test cases
- It has a beneficial side effect in pointing out incompleteness and ambiguities in specifications
- **Steps:**
  - Identify the causes and effects from the specification
  - Develop the cause effect diagram
  - Create a decision table
  - Develop test cases from the decision table



Copyright © Capgemini 2015. All Rights Reserved 60

BVA and Equivalence partitioning do not explore combinations of input circumstances.

In Cause effect graphing, causes and effects are identified.

Cause is a distinct input condition. Effect is a distinct output condition.

Cause and Effect - Simple example

E.g. 1

Cause: Got caught in rain

Effect: Cold and cough

E.g. 2

Cause: Hours of Dance practice

Effect: First Prize in the competition

2.1: Dynamic Testing

## Cause Effect Graphing

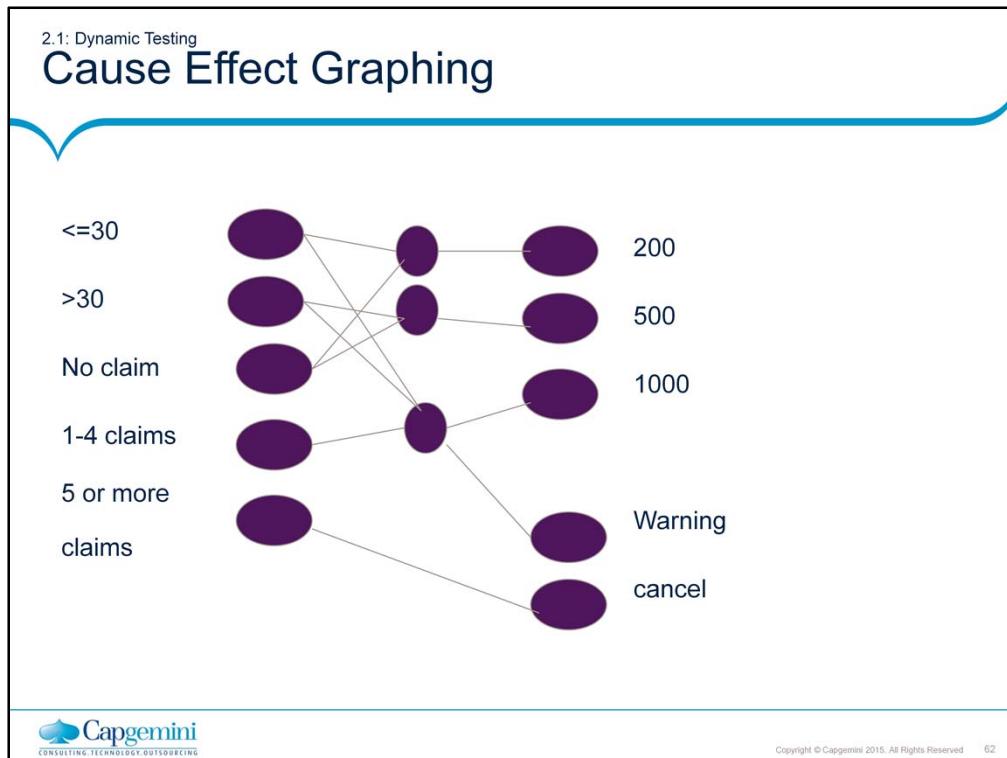
### ▪ Insurance policy renewal example

- An insurance agency has the following norms to provide premium to its policy holders
- If age<=30 and no claim made premium increase will be 200 else 500
- For any age if claims made is 1 to 4 then premium increase will be 1000
- If one or more claims made then warning letter, if 5 or more claims made then cancel policy



Copyright © Capgemini 2015. All Rights Reserved 61

It helps to create the cause and effect diagram through the decision table.



The above graph covers all the combinations of the scenarios where

1. If age $\leq 30$  and no claim made premium increase will be 200 else 500
2. For any age if claims made is 1 to 4 then premium increase will be 1000
3. If one or more claims made then warning letter, if 5 or more claims made then cancel policy

2.1: Dynamic Testing

## Cause Effect Graphing

▪ Insurance Renewal Decision Table

| No. of Claims | Insured Age | Premium Increase | Send Warning | Cancel   |
|---------------|-------------|------------------|--------------|----------|
| 0             | <=30<br>>30 | 200<br>500       | No<br>No     | No<br>No |
| 1-4           | All ages    | 1000             | Yes          | No       |
| 5 or more     |             |                  | No           | Yes      |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 63

The cause effect graph is converted into decision table. The entire policy terms get transformed into tabular format in more easier way.

1. If age<=30 and no claim made premium increase will be 200 else 500. No warning will be sent and no cancellation is done.
2. For any age if claims made is 1 to 4 then premium increase will be 1000. Warning will be sent but no cancellation is done
3. If 5 or more claims made then cancel policy

2.1: Dynamic Testing

## Cause Effect Graphing

| Causes        | Effects                     |
|---------------|-----------------------------|
| C1 age<=30    | E1 Premium increase 200     |
| C2 age >=30   | E2 Premium increase 500     |
| C3 No claims  | E3 Premium increase by 1000 |
| C4 1-5 claims | E4 Warning Letter           |
| C5 >5 claims  | E5 Cancel Premium           |



Copyright © Capgemini 2015. All Rights Reserved 64

Policy terms are mentioned in the cause and effect manner. All the conditions are termed as causes and its result is reflected as effects in the above table.

2.1: Dynamic Testing

## State Transition Testing

- A testing techniques that aids to validate various states when a program moves from one visible state to another.

Menu System Example :

- The program starts with an introductory menu. As an option is selected the program changes state and displays a new menu. Eventually it displays some information , data input screen.
- Each option in each menu should be tested to validate that each selection made takes us to the state we should reach next.



Copyright © Capgemini 2015. All Rights Reserved 65

For Example:

A registration form has two buttons, viz. OK and CANCEL. After filling up the entire application, OK button changes its caption to SAVE to save the filled data.

So the single button is acting in two different ways depending on its state transition, which has to be tested.

2.1: Dynamic Testing

## State Transition Testing

- Washing machine has different modes like soak, wash, rinse & dry
- Machine in these different states, exhibit different features
  - Soak mode - clothes absorb soap water
  - Wash mode - clothes get washed with soap water
  - Rinse mode - It removes soap water from clothes
  - Dry mode - water gets removed from clothes
- It is useful to create a state transition diagram to spot relationship between states and trace transition between states



Copyright © Capgemini 2015. All Rights Reserved 66

### Example :

Drawn above is a classic example of State Transition testing. It depicts about the Stack implementation.

1. Initially Stack is in **Empty** state.
2. As soon as, some elements get added to it with push() method, its state changes to **Loaded**.
3. When the element reaches to the Stack's maximum capacity, its state changes from **Loaded** to **FULL**.
4. Similarly, while removing or accessing from the Stack with pop() method, it gets the element into **Loaded** state. It extracts the topmost element.

2.2: Test Case Construction/ Test Data Preparation

## Introduction

- Test cases construction and test data preparation are the first stages of testing
- Test cases are prepared based on test ideas
- “A test idea is a brief statement of something that should be tested.”
  - For example, if you’re testing a square root function, one idea for a test would be ‘test a number less than zero’
- “The idea of preparing a test case is to check if the code handles an error case.”

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 67

Designing good test cases is a complex art. The complexity comes from three sources:

- Test cases help us discover information. Different types of tests are more effective for different classes of information.
- Test cases can be “good” in a variety of ways. No test case will be good in all of them.
- People tend to create test cases according to certain testing styles, such as domain testing or risk-based testing. Good domain tests are different from good risk-based tests.

## 2.2: Test Case Construction

## Test Case

- A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- Test cases may be designed based on -
  - Values – Valid/Invalid/Boundary/Negative
  - Test conditions
- Test case will be complex if there is more than one expected result.



Copyright © Capgemini 2015. All Rights Reserved 68

A test case is a question that we ask the program. The point of running the test is to gain information, for example whether the program will pass or fail the test.

**Characteristics of a Good Test:**

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

2.2.2: Test Case Construction

## Test Case Terminologies

- **Test Scenario**

- It is an end-to-end flow of a combination of test conditions & test cases integrated in a logical sequence, covering a business processes

- **Test Condition / Pre Condition**

- Environmental and state which must be fulfilled before the component/unit can be executed with a particular input value.
  - It is a set of rules under which a tester will determine if a requirement is partially or fully satisfied
  - One test condition will have multiple test cases



Copyright © Capgemini 2015. All Rights Reserved 69

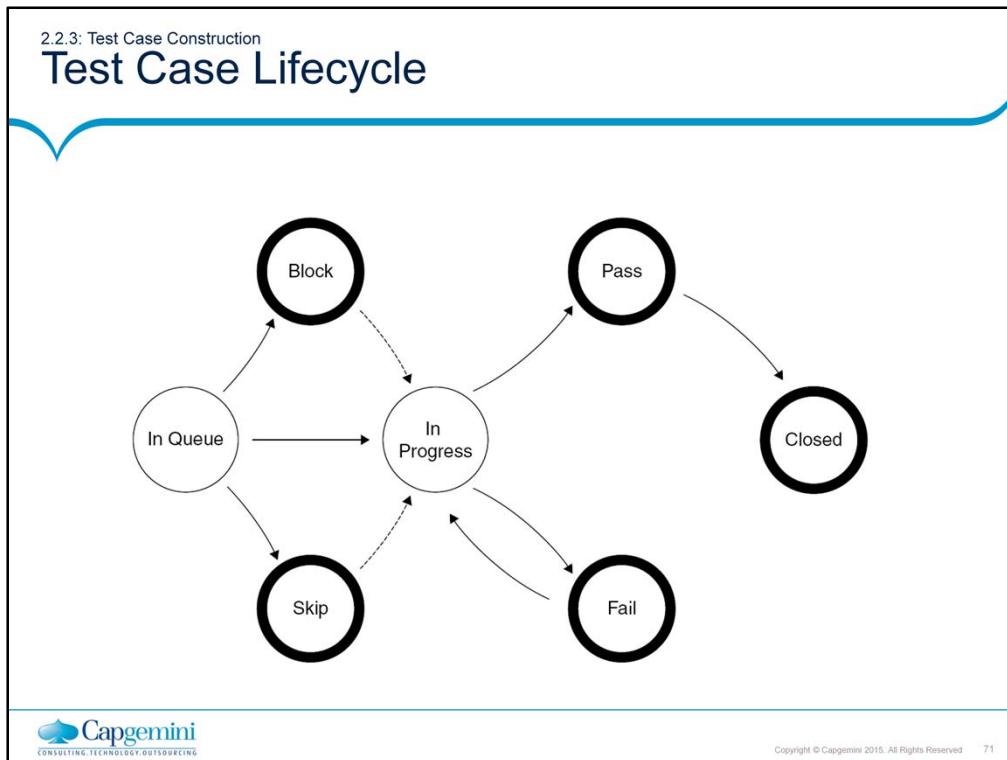
2.2.2: Test Case Construction

## Test Case Terminologies

- Test Data/Input
  - Inputs & its combinations/variables used
- Test Procedure
  - A detailed description of steps to execute the test
- Test Result/Output
  - Pass / Fail – If the program works as given in the specification, it is said to Pass otherwise Fail.
  - Failed test cases may lead to code rework



Copyright © Capgemini 2015. All Rights Reserved 70



2.2.3: Test Case Construction

## Test Case Lifecycle

- Write an effective and efficient set of test cases.
- The set of tests should find common bugs.
- The set of tests finds those bugs with a reasonable amount of effort.
- Test case incorporated tester action, data, and expected result. The action is usually to input some data, but it might be other actions as well.
- A template is usually followed for writing test cases.



Copyright © Capgemini 2015. All Rights Reserved 72

2.2.3.2: Test Case Construction

## Test Case Design Technique

- Test cases are designed based on the following techniques
  - Specification-based - Black Box testing techniques
    - Boundary value analysis, Equivalence partitioning, decision table
  - Structure – based – White Box testing techniques
    - Code coverage, decision coverage, statement coverage
  - Experience based techniques
    - Exploratory testing, fault attack



Copyright © Capgemini 2015. All Rights Reserved 73

## 2.2.3.2: Test Case Construction

# Test Case Template



Copyright © Capgemini 2015. All Rights Reserved

**Note:** Test case template may vary from project to project. Based on project and client requirement few additional fields can be added in test case template like test environment, module/functionality column/test case type column etc.

2.2.3.2: Test Case Construction

## Test Case Example

- Writing a testing case to test the following
  - A login form takes user name and password data as input in two text boxes
  - Checks for valid user name and password
  - Enters home page if user name and password is valid
  - Else it displays error message

Login Page

|                                      |                          |
|--------------------------------------|--------------------------|
| Username                             | <input type="text"/>     |
| Password                             | <input type="password"/> |
| <input type="button" value="Login"/> |                          |



Copyright © Capgemini 2015. All Rights Reserved 75

The format of test case for login screen should contain :

**1. +ve test cases for login.**

The positive test case can test with valid user name and password as input data.

**2. -ve test cases for login.**

In the -ve scenario test case should cover all invalid combinations of input data such as

- User name is null
- Password is null
- Both username and password are null
- Invalid username
- Invalid password

## 2.2.3.2: Test Case Construction

## Test Case Example

- Test Preconditions:

- User should have valid username and password
  - All the fields must be enabled
  - Application should have proper connection with the database

- Assumptions:

- If there any assumptions while writing the test cases that can be written in the assumption field or column.

- Example:

- Username: Jim
  - Password: pass123
  - Username: Harry
  - Password:mypass123



Copyright © Capgemini 2015. All Rights Reserved 76

2.2.3.2: Test Case Construction

## Test Case Example

- **Test Post Conditions:**

- The user is either logged in and home page is displayed
  - Error message is displayed if either the username or password is incorrect

- **Test Objective:**

- To check whether the entered username and password are valid or invalid
    - Home page is displayed for valid inputs
    - Error message is displayed for invalid inputs



Copyright © Capgemini 2015. All Rights Reserved 77

2.2.3.2: Test Case Construction

## Test Case Design Technique

Assumptions: Valid UserName and password are available in database

|  | Test Case ID | Preconditions (If any)                                     | Test Condition / Scenario   | Test Steps         | Input /Test Data | Expected Result  | Actual Result |
|--|--------------|--|---|--------------------|------------------|--|---------------|
| The application is invoked and login page is displayed | TC_01        | User name and password field in the login form are enabled | To validate the login page with null password                       | Enter User Name    | Jim              |  |               |
|  |              |  |   | Enter password     |                  |  |               |
|  |              |  |   | Press LOGIN Button |                  | Should Display Warning Message Box "Please Enter User name and Password" |               |
| The application is invoked and login page is displayed | TC_02        | User name and password field in the login form are enabled | To validate the login page with null username                       | Enter User Name    |                  |  |               |
|  |              |  |   | Enter password     | igate123         |  |               |
|  |              |  |   | Press LOGIN Button |                  | Should Display Warning Message Box "Please Enter User name and Password" |               |
| The application is invoked and login page is displayed | TC_03        | User name and password field in the login form are enabled | To validate the login page with valid username and invalid password | Enter User Name    | Harry            |  |               |
|  |              |  |   | Enter password     | mypass123        |  |               |
|  |              |  |   | Press LOGIN Button |                  | Should Display Warning Message Box "Please Enter User name and Password" |               |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 78.

| 2.2.3.2: Test Case Construction                        |       |  |   |                    |         |  |
|--|-------|--|---|--------------------|---------|--|
| Test Case Design Technique                             |       |  |   |                    |         |  |
| The application is invoked and login page is displayed | TC_04 | User name and password field in the login form are enabled | To validate the login page with invalid username and valid password   | Enter User Name    | XYZ     |  |
|  |       |  |   | Enter password     | pass123 |  |
|  |       |  |   | Press LOGIN Button |         | Should Display Warning Message Box "Please Enter User name and Password" |
| The application is invoked and login page is displayed | TC_05 | User name and password field in the login form are enabled | To validate the login page with invalid username and invalid password | Enter User Name    | XYZ     |  |
|  |       |  |   | Enter password     | XYZ     |  |
|  |       |  |   | Press LOGIN Button |         | Should Display Warning Message Box "Please Enter User name and Password" |
| The application is invoked and login page is displayed | TC_06 | User name and password field in the login form are enabled | To validate the login page with null username and null password       | Enter User Name    |         |  |
|  |       |  |   | Enter password     |         |  |
|  |       |  |   | Press LOGIN Button |         | Should Display Warning Message Box "Please Enter User name and Password" |
| The application is invoked and login page is displayed | TC_07 | User name and password field in the login form are enabled | To validate the login page with valid username and password           | Enter User Name    | Jim     |  |
|  |       |  |   | Enter password     | pass123 |  |
|  |       |  |   | Press LOGIN Button |         | Should navigate to the Home page   |



Copyright © Capgemini 2015. All Rights Reserved 79

2.2.4: Test Data Preparation

## What is test data?

### ■ Test Data

- An application is built for a business purpose. We input data and there is a corresponding output. While an application is being tested we need to use dummy data to simulate the business workflows. This is called test data.
- A test scenario will always have an associated test data. Tester may provide test data at the time of executing the test cases or application may pick the required input data from the predefined data locations.
- The test data may be any kind of input to application, any kind of file that is loaded by the application or entries read from the database tables. It may be in any format like xml test data, stand alone variables, SQL test data etc.
- If you are testing with bad or unstable data, how can you be sure your test results are accurate!!!



Copyright © Capgemini 2015. All Rights Reserved 80

## 2.2.4: Test Data Preparation

## Test Data team

- Test Data team should have Data Coordinator and team members. Test data teams are structured in various different ways.
  - Dedicated test data team
  - Development team as a test data team
  - QA team as a test data team
- Data Coordinator's role and responsibility - Data coordinator will be the point of contact between the main stakeholders. He will be responsible for gathering all data requirements.
  - Documentation of knowledge of interfaces and test data, mentoring and advising test team on data use, and support on the end-to-end flow;
  - Data Coordinator will be responsible of the data prioritization, according to the timelines fixed by data team for executing and delivering the requested data.



Copyright © Capgemini 2015. All Rights Reserved 81

Self explanatory

## 2.2.4: Test Data Preparation

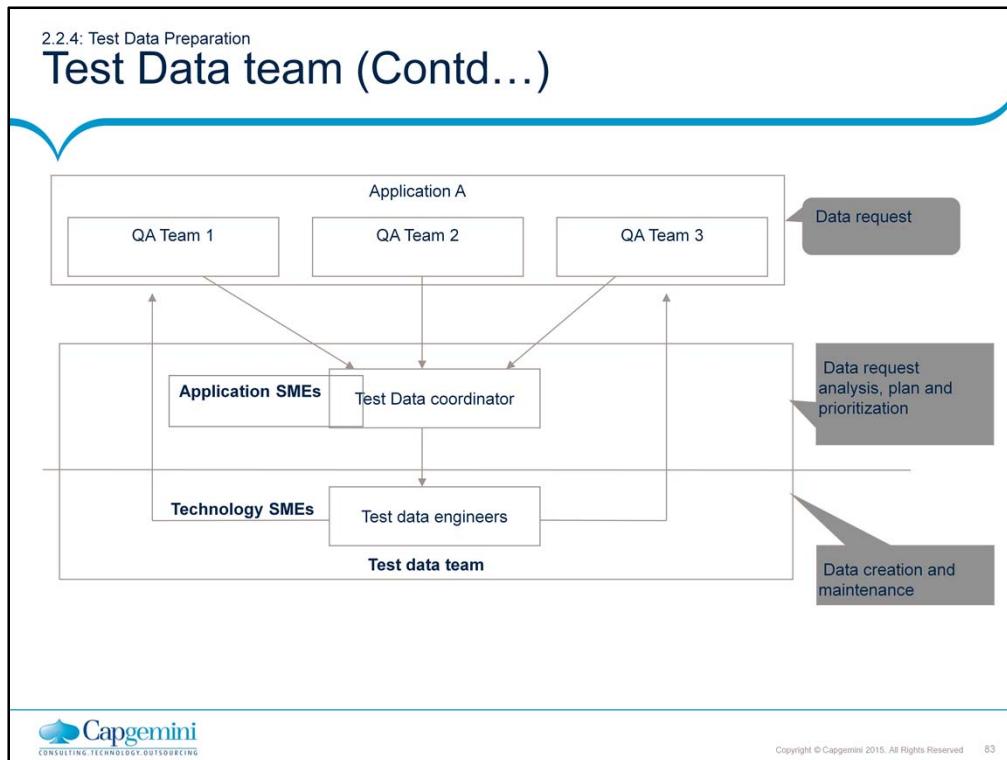
## Test Data team (Contd...)

- Participation in test case planning, collection and providing data to support test cases for development and execution
- Providing help in handling, managing and manipulating test data
- Data Coordinator must ensure the correct application of masking rules, since each test case can have a set of static data necessary for execution
- Test Data Engineer's role and responsibility – Test data engineer will be responsible to create the data as per the requirement . He should be doing following activities during data creation
  - Understand the Requirements
  - Understand the DB and Table structures of the applications in case data generation for database testing
  - Understand the volume of data required
  - Automate the process of data generation if volume is huge

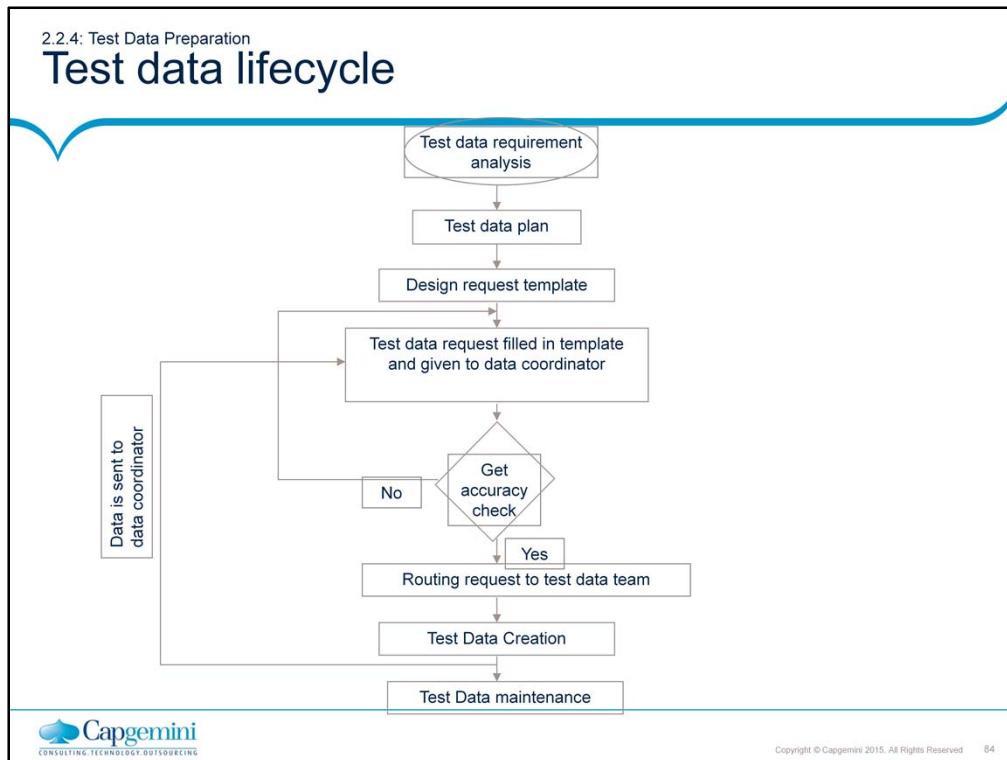


Copyright © Capgemini 2015. All Rights Reserved 82

Self explanatory



Self explanatory



The way we have SDLC and STLC for software, the same way test data has to follow a defined path.

It having similar phases as STLC:

1. Requirement
2. Plan
3. Creation
4. Usage and maintenance

2.2.4: Test Data Preparation > Test data life cycle

## Requirement and Planning

- Test data requirement analysis
  - Test data analysis is the most important part of test data management. While preparing the test cases and test scripts, the test analyst needs to understand the exact data requirement and needs to parameterize the same
  - Parameterizing the test data requirement for the test cases would reduce the effort in understanding the test data requirement during the data mapping phase
  - Test data might be converted data from the legacy system. At this point initial test data analysis and parameterization would help to identify those data requirements that can be picked from legacy system and which need to create from scratch
- Once test case design is over test data requirement should also be completed.
- Traceability matrix will be generated to ensure complete coverage of requirements.



Copyright © Capgemini 2015. All Rights Reserved 85

Parameterizing means a value or a symbolic reference to a value. Give some value for each data requirement. So If sample is available then its easy to understand the exact data requirements.

2.2.4: Test Data Preparation &gt; Test data life cycle

## Requirement and Planning (Contd...)

- Test data planning – Test data planning includes following:
  - Test data requirement template finalization
  - Defining the request initiation and completion process
  - Data organization
  - Data creation tools and technologies
  - Test data state control mechanism
  - Data maintenance planning



Copyright © Capgemini 2015. All Rights Reserved 86

Planning -> A standard template will always be helpful during requirement gathering as all the users have the common understanding of the fields in template.

By looking at the requirements need to plan if there is any tool is required to generate the data. For example any kind of database testing will be required millions of records that are not possible to create manually.

Once data is created it might be required for subsequent iterations or releases, in that case data need to maintain.

For maintenance it's required to know the data state after the execution and the required state for next iteration. So proper data state control mechanism should be in place.

2.2.4: Test Data Preparation > Test data life cycle

## Request Process

- The communication between data coordinator and QA Testing Team must follow certain standards in terms of requests template. The data coordinator will act as a single entry point for the QA Team and will be able to retrieve data for testing.
- Data request Template
  - The Testing Team will address their requests to the data coordinator through a standard template
  - The template includes fields to fulfil with detailed information's for each kind of requested data. The tester has to describe very precisely the needs of required data for execution of test cases to ensure quick and consistent delivery of data from the data team
  - The fields integrated in the request template are grouped for each kind of data
  - The QA Testers have to fill the template and when it's possible add a sample of required data
  - The data coordinator will be in charge of the data request template from QA team



Copyright © Capgemini 2015. All Rights Reserved 87

2.2.4: Test Data Preparation > Test data life cycle

## Request Process (Contd...)

### ▪ Request Accuracy check

- After the receipt of the request, a manual accuracy check is executed by the data Coordinator on the following points:
  - Validate the template used by the users whether it's standard one or not
  - All mandatory fields are filled in and filled in properly
  - Any duplicated request or incomplete request templates
  - Requests of data already existing (extracted) in Data Base
  - The data coordinator will provide the reason for having rejected the template and then acknowledge the requestor by email
  - In case the request file from the tester passes the Accuracy Check, then the data coordinator will move that request to data creation team



Copyright © Capgemini 2015. All Rights Reserved 88

Give example of existing project.

## 2.2.4: Test Data Preparation

## Test Data creation techniques

- There are two ways to create the test data.
  - Data from scratch – Any new development project, there will be no production data to mimic. So there is a need of data creation from scratch. Though this approach is free of all the privacy issues, still it is very difficult to actually come up with test data easily that actually mimics production environment
  - While designing the test data following categories need to consider:
    - **No data:** In case of no input data proper error message should pop up.
    - **Valid data set:** Data is created to check whether required behavior is exhibit if data given is a valid data
    - **Invalid data set:** Check for negative values
    - **Illegal data format:** Make one data set of illegal data format. System should not accept data in invalid or illegal format. Also check proper error messages are generated



Copyright © Capgemini 2015. All Rights Reserved 89

An application accepting integer values (that is whole number values) between -10,000 to +10,000 can be expected to be able to handle negative integers, zero and positive integers. Therefore, the set of input values can be divided into three partitions:

From -10,000 to -1, 0 and from 1 to 10,000

## 2.2.4: Test Data Preparation

## Test Data creation techniques (Contd...)

- Boundary Condition data set: Data set containing out of range data. Identify application boundary cases and prepare data set that will cover lower as well as upper boundary conditions
- Equivalent partition: Set of data to an input condition that give the same result when executing a program and partitioning them from another equivalent set of data for the same condition
- Decision Tree: This model is used to visually represent the paths from input to all possible outputs of a given system. Data is created for each node and corresponding branches
- State transition: State-Transition testing is identifying the states, events, actions, and transitions that should be tested. It also define how a system interacts with the outside world, the events it processes

2.2.4: Test Data Preparation

## Test data from production data

- Test data from production data

- The typical approach for data creation is using old production data or by transforming production data so as to replace actual values with equivalent values
- Production data can't be used as it is. Most of the time it's a sensitive information like credit card number etc. So masking is done over the production data to protect sensitive information
- After sanitization, the database remains perfectly usable - the look-and-feel is preserved - but the information content is secure



Copyright © Capgemini 2015. All Rights Reserved 91

For example any search engine project 'statistics testing'. To check history of user searches and clicks on advertiser campaigns large data was processed for several years which is practically impossible to manipulate manually for several dates spread over many years. So there is no other option than using live server data backup for testing.

## 2.2.4: Test Data Preparation

## Test data life cycle - Maintenance

- Data maintenance

- Data maintenance is a sizeable task, and it is be substantial fraction of overall test maintenance costs

- Activities

- Data maintenance team should closely with the execution team to monitor any data related defects that are raised during the execution phase
  - In case of data corruption due to defect fixes or due to major deployment, the test data management team should take the lead and analyze the extent of data corruption and start data mapping activity as a part of maintenance

2.2.4: Test Data Preparation

## Test data life cycle - Maintenance (Contd...)

- Other data maintenance activates include:
  - Replacing non-reusable data for regression testing
  - Data Manipulation
  - Responding to change - database schema, code, requirements
  - New test requirements
- **BEST PRACTICES**
  - Data requirement analysis as a part of preparation activity
  - Parameterize the data requirement of a test case/test script
  - Split the test cases as Data Reusable and Data Non-reusable
  - During execution if There are data defects raise the same as a defect in a tool and assign it to the data maintenance team
  - Test data maintenance should be a parallel activity along side execution since quality of data determines the quality of our testing



Copyright © Capgemini 2015. All Rights Reserved 93

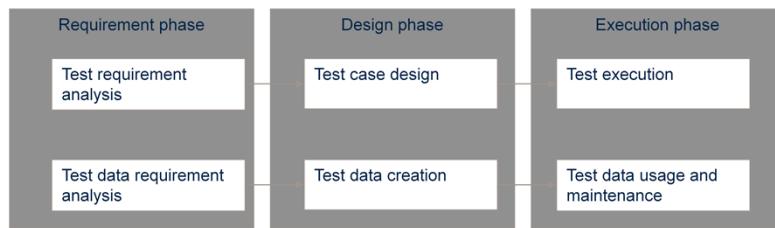
**Data-Reusable Test case:** A test case or a test script that does not change the nature of the test data that has been mapped to it. An example would be customer enquiry test cases, where customer identification number (CIN) or an account number (A/c) is mapped to the test case. While executing the test case, the data does not get corrupt. Hence this data can be reused. Such test cases are classified as Data-Reusable test cases.

**Data-Non-Reusable:** A test case or a test script that changes the nature of the test data that has been mapped to it comes under this category. An example would be a case where a customer is marked as deceased. Once data is modified after the execution of the test script, the same data cannot be reused across another test case.

2.2.4: Test Data Preparation

## Test data in STLC - Staggered with test case Design

| +ve   | -ve   |
|---|---|
| <ul style="list-style-type: none"> <li>All Test data will be ready before Test Execution phase.</li> <li>As Test cases are designed and signed-off, data creation will happen almost concurrently.</li> </ul> | <ul style="list-style-type: none"> <li>There are issues such as Resource overloading, idle time due to lesser test case inputs or not having logical test cases, which may lead to load balancing challenges</li> </ul> |

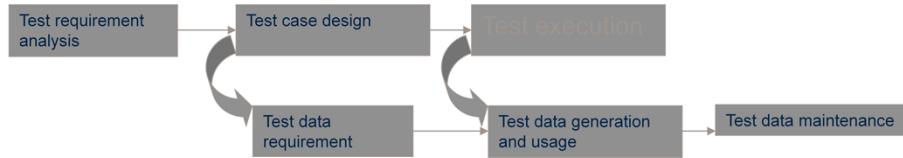


Copyright © Capgemini 2015. All Rights Reserved 94

Self explanatory

2.2.4: Test Data Preparation

## Test data in STLC - Staggered with test case Execution



| +ve   | -ve  |
|---|--|
| <ul style="list-style-type: none"> <li>• Test data creation will happen as per Test Execution Priority</li> <li>• Minimizes rework as all Test Cases will be designed and signed-off prior to Test Data creation</li> </ul> | <ul style="list-style-type: none"> <li>• Test data dependencies between different areas to be tested might be overlooked</li> <li>• Some of the Test data creation can be redundant</li> </ul> |



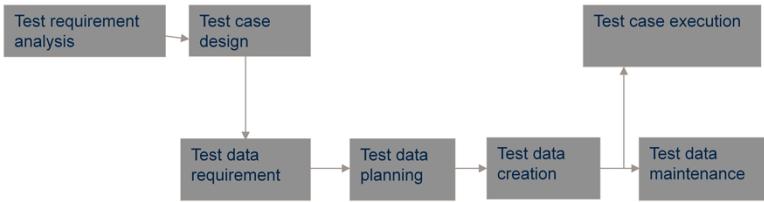
Copyright © Capgemini 2015. All Rights Reserved 95

Self explanatory

2.2.4: Test Data Preparation

## Test data in STLC -Standalone phase between Test Case Design and Test Case Execution

| +ve  | -ve   |
|--|---|
| <ul style="list-style-type: none"> <li>• Test data creation happens with clear exit criteria from Test design phase and with clear entry criteria for the Test Execution phase</li> <li>• Data creation will be completed so There will be no or minimal resource load balancing problems</li> </ul> | <ul style="list-style-type: none"> <li>• If not planned in advance, Test Data creation activities can delay the start of Test execution activities</li> </ul> |



Copyright © Capgemini 2015. All Rights Reserved 96

Self explanatory

2.3. Positive Test

## What is positive testing?

- Positive testing can be performed on the system by entering the valid data as input. It checks whether an application behaves as expected with the positive input
- By using positive test we can check the application that does what it is supposed to do
- When tester test the application from positive point of mind then it is known as positive testing
- Testing aimed at showing software works. Also known as “test to pass”
- Positive testing, many times referred to as “Happy path testing” is generally the first form of testing that a tester would perform on an application



Copyright © Capgemini 2015. All Rights Reserved 97

**Note:** Equivalence Partitioning and boundary value analysis are test case design techniques used for writing positive test cases.

2.3 Positive Test

## Example of positive testing

- Example of positive testing
- Consider a scenario where you want to test an voting application which contains a simple textbox to enter age and requirement is that it should take only integers values and the value should be greater than 18. So here provide only positive integer values which are greater than 18 to check whether it is working as expected or not is the Positive Testing. Most of the applications developers are implement Positive scenarios. Testing point of view testers get less defects count around positive testing.

&gt; 18      19

Age:

Enter only integer values



Copyright © Capgemini 2015. All Rights Reserved 98

2.3. Positive Test

## Advantages/Limitations of positive testing

- Advantages of positive testing

- Positive testing proves that a given product and project always meets the requirements and specifications
- Positive testing is useful in the normal day to day life scenarios as it checks the expected behavior of application
- Positive testing ensures that the business use case is validated

- Limitations of Positive testing:

- Positive tests check for only valid set of values



Copyright © Capgemini 2015. All Rights Reserved 99

## 2.3. Positive Test

## Positive test scenarios

- Let's take example of Positive testing scenarios:
  - If the requirement is saying that password text field should accept 5 – 15 characters and only alphanumeric characters.
- Positive Test Scenarios:
  - Password textbox should accept 5 characters
  - Password textbox should accept upto 15 characters
  - Password textbox should accept any value in between 5-15 chars length
  - Password textbox should accept all numeric & alphabets values

## 2.4. Negative Tests

## What is negative testing?

- Negative Testing can be performed on the system by providing invalid data as input. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do
- When tester/User test the application from negative point of mind then it is known as negative testing
- Negative testing is the process of applying as much creativity as possible and validating the application against invalid data
- The purpose of Negative testing is to break the system and to verify the application response during unintentional inputs
- Testing aimed at showing software does not work. Also known as “test to fail”



Copyright © Capgemini 2015. All Rights Reserved 101

**Note:** Equivalence Partitioning and boundary value analysis are test case design techniques used for writing negative test cases.

## 2.4. Negative Tests

## Example of negative testing

- Example of Negative testing:

- In the voting application scenario, Negative testing can be performed by testing by entering alphabets characters from A to Z or from a to z. Age text box should not accept the values or it should throw an error message for these invalid data inputs.

values > 18

Age:

Daya223

Enter only integer

## 2.4. Negative Tests

## Advantages/Limitations of negative testing

- Advantages of Negative Testing:
  - Negative testing helps to improve the testing coverage of your software application under test
  - Negative testing discovers 'hidden' errors from application under test
  - Negative testing help to find more defects & improve the quality of the software application under test
  - negative testing ensures that the delivered software has no flaws
- Limitation of Negative Testing
  - Negative tests check for only invalid set of values



Copyright © Capgemini 2015. All Rights Reserved 103

## 2.4.Negative Tests

## Negative test Scenarios

- Let's take example of Negative testing scenarios:
  - If the requirement is saying that password text field should accept 5 – 15 characters and only alphanumeric characters.
- Negative Test Scenarios:
  - Password textbox should not accept less than 5 characters
  - Password textbox should not exceed more than 15 characters
  - Password textbox should not accept special characters



Copyright © Capgemini 2015. All Rights Reserved 104

**Note: Requirements:**

The ID has to be a number between 1- 200

The ID is mandatory.

Write some positive and negative test scenarios for this particular pane.

## 2.5. Basic Tests

## What is basic test?

- Basic tests are used to test very basic functionality of software
- The basic tests also verifies end to end builds
- Basic test are always positive tests
- Basic test can be smoke test or sanity test

## 2.5 Basic Tests

## Example on Basic test

- Customer Relationship Management (CRM) application is business philosophy towards customers. To focus on their needs and improve customer relationships, with view to maximize customer satisfaction. So, in CRM application customer creation is basic functionality that should work. So the basic test focus is on Login and then customer creation. The basic test for this CRM application is Customer login and then customer creation with mandatory fields.

2.6. Alternate Tests

## What is alternate test?

- Sometimes there maybe more than one way of performing a particular function or task with an intent to give the end user more flexibility or for general product consistency
- This is called alternate testing
- Alternate test is a kind of positive testing
- In alternate path testing the test is again performed to meet its requirements but using different route than the obvious path
- The test scenario would even consume the same kind of data to achieve the same result

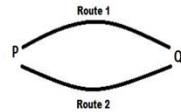


Copyright © Capgemini 2015. All Rights Reserved 107

## 2.6. Alternate Tests

## Example on alternate test

- Alternate test



- P is a starting point and Q is the end point. There are two ways to go from P to Q. Route 1 is the generally taken route and Route 2 is an alternative route. Therefore in such a case, happy path testing would be traversing from point P to Q using Route 1 and the alternate test would comprise taking Route 2 to go from P to Q. Observe that the result in both the cases is the same.

2.7.Importance of writing positive,negative,basic,alternate test while designing test cases.

## Importance of writing positive, negative, basic, alternate test while designing test cases

- Approach of writing positive, negative, basic, alternate test are useful to design effective test cases which help to improve quality of software
- These approach to test case design are help to improve the test case design coverage
- By using these approach test cases are written for real life scenarios. It ensures real life scenarios are tested before moving software live
- By designing positive and negative test cases ensures that the application works as per the requirements and specifications
- By executing effective test cases, helps to find more defects before releasing software, so it builds confidence in system



Copyright © Capgemini 2015. All Rights Reserved 109

## Summary

- In this lesson, you have learnt:
  - The test case techniques discussed so far need to be combined to form overall strategy
  - Each technique contributes a set of useful test cases, but none of them by itself contributes a thorough set of test cases



## Review Question

- Question 1: \_\_\_\_\_ testing can discover dead codes
- Question 2: The objective of walkthrough is to find errors but not solutions
  - Option: True / False
- Question 3: For calculating cyclomatic complexity, flow graph is mapped into corresponding flow chart
  - Option: True / False
- Question 4: How many minimum test cases required to test a simple loop?



## Review Question

- Question 5: One test condition will have \_\_\_\_\_ test cases
- Question 6: For Agile development model conventional testing approach is followed
  - Option: True / False
- Question 7: A test case is a set of \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ developed for a particular objective



## Review Question: Match the Following

1. Code coverage

2. Interface errors

3. Code complexity

A. Flow graph

B. Loop testing

C. Black box testing

D. Flow chart

E. Condition testing

F. White box testing



# **Testing Concepts for V&V Automation Testing**

Lesson 3: Types of System  
Testing

## Lesson Objectives

- To understand the following topics:
  - Verification and Validation
  - SDLC & V Model
  - Testing phases
  - Types of system testing
  - Acceptance testing- Alpha testing, Beta testing
  - Exploratory testing
  - Test strategy



3.1: V&V

## Verification and Validation

- Verification
  - Verification refers to a set of activities which ensures that software correctly implements a specific function.
  - Purpose of verification is to check: Are we building the product right?
  - Example: code and document reviews, inspections, walkthroughs.
  - It is a Quality improvement process.
  - It is involve with the reviewing and evaluating the process.
  - It is conducted by QA team.
  - Verification is Correctness.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

V&V encompasses many of the activities that are encompassed by S/w quality assurance that include formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, installation testing.

Example of Verification : code and document inspections, walkthroughs, and other techniques. unit testing , integration testing , system testing

If we are in a shopping centre and buy a thing with a code number 2342 and when we go to till and they check the number of that item and find it wrong then system will check all product number of the relevant number but don't find any number of this kind then we can say that the verify thing is wrong.

Verification is a process, which performs testing to ensure implemented functions meeting to designed functions.

3.1: V&V

## Verification and Validation

- Validation
  - Purpose of Validation is to check : Are we building the right product?
  - Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
  - After each validation test has been conducted, one of two possible conditions exist:
    - 1. The function or performance characteristics conform to specification and are accepted, or
    - 2. Deviation from specification and a deficiency list is created.
  - Example : a series of black box tests that demonstrate conformity with requirements.
  - It is ensures the functionality.
  - It is conducted by development team with the help from QC team.
  - Validation is Truth.
  - Validation is the following process of verification.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

Validation of software typically includes evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements.

3.2: SDLC and V Model

## Why Write Test Cases Before Coding?

- When adding a new feature or enhancing an existing solution, writing test cases forces you to think about what the code is supposed to accomplish.
- You end up with a clean and simple design that does exactly what you expect it to do.



Copyright © Capgemini 2015. All Rights Reserved 5

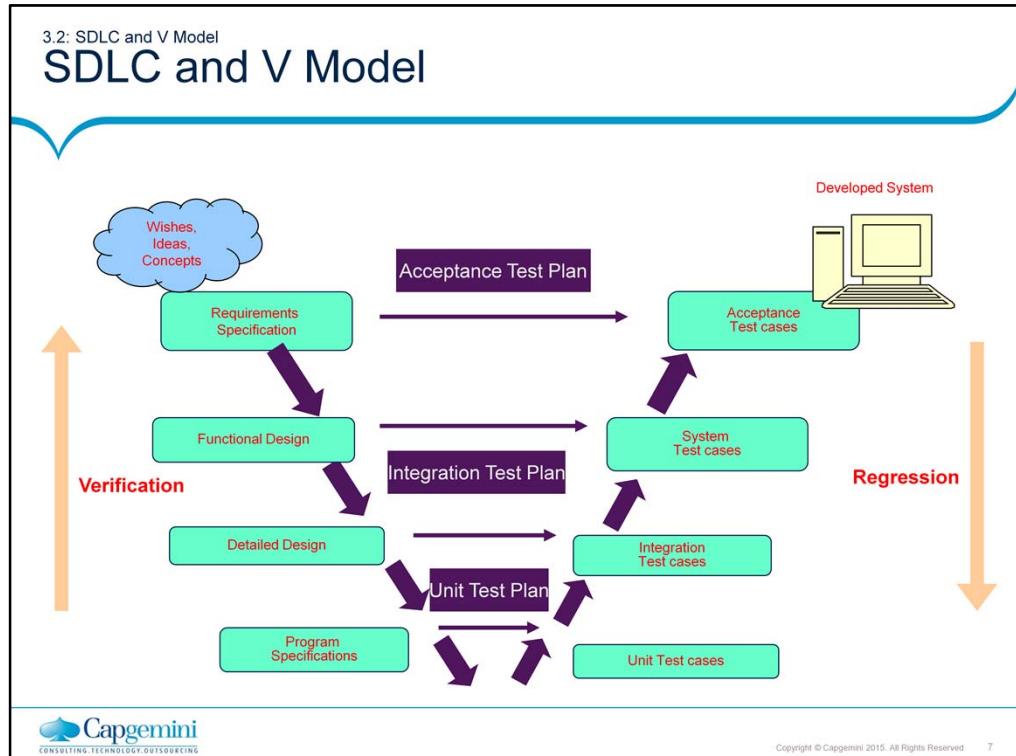
3.2: SDLC and V Model

## Introduction

- There are some distinct test phases that take place in each of the software life cycle activity.
- It is easier to visualize through the famous Waterfall model of development and V- model of testing.
- The V proceeds from left to right, depicting the basic sequence of development and testing activities.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6



Briefly discuss the Software Development Life cycle activities and the various Models that are used like the Waterfall, RAD, Spiral ,XP etc

Then concentrate on the waterfall model above.

The four main process phases – **requirements, specification, design and implementation**

- They have a corresponding **verification and validation testing** phase
- **Implementation** of modules is tested by **unit testing**.
- **System design** is tested by **integration testing**.
- **System specifications** are tested by **system testing**.
- **Acceptance testing** verifies the **requirements specification**.
- Starting from the requirements, the system is developed one phase at a time until the lowest phase, the implementation phase, is finished.
- At this stage testing begins, starting from unit testing and moving up one test level at a time until the acceptance testing phase is completed

3.2: SDLC and V Model

### V Model

- The model is valuable because it highlights the existence of several levels or phases of testing and depicts the way each relates to a different development phase.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 8

3.3: Testing phases

## Testing Phases

- Unit testing
  - Unit testing is code-based and performed primarily by developers to demonstrate that their smallest pieces of executable code function suitably.
- Integration testing
  - Integration testing demonstrates that two or more units or other integrations work together properly, and tends to focus on the interfaces specified in low-level design.
- System testing
  - 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.
- Acceptance testing
  - Acceptance testing is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

3.4: Unit testing

## Introduction

- 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

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

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.

3.4: Unit testing

## Unit/Module Testing

- Unit testing uncovers errors in logic and function within the boundaries of a component.

Local data structures  
Boundary conditions  
Independent paths  
Initialization , Loops, Control flow errors  
Computations, Comparison,  
Error handling paths

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

The module interface is tested to ensure that information properly flows into and out of the program under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.

Test cases should uncover errors such as

- (1) comparison of different data types
- (2) incorrect logical operators or precedence
- (3) expectation of equality when precision error makes equality unlikely
- (4) incorrect comparison of variables
- (5) improper or nonexistent loop termination
- (6) failure to exit when divergent iteration is encountered.

3.5: Integration testing

## Introduction

- Testing of combined parts of an application to determine if they function together correctly
- The main three elements are interfaces, module combinations and global data structures
- Attempts to find discrepancies between program & its external specification (program's description from the point of view of the outside world)
- Testing a module to check if the component of the modules are integrated properly is example of integration testing

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

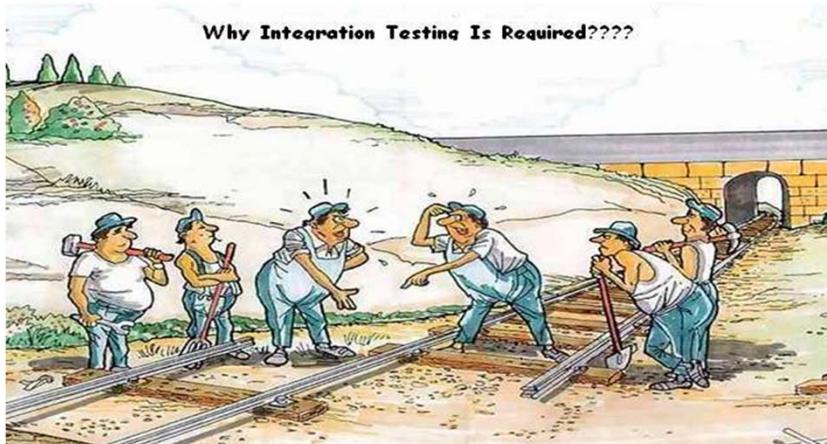
Interfaces are the means by which data is passed to and from modules. “Interface integrity” – to ensure that when data is passed to another module, by way of a call, none of the data becomes lost or corrupted. This loss or corruption can happen by number of ways- calling and receiving parameters may be of the wrong type and so the data appears in the receiving programs in a garbled form; there may be different number of calling and receiving parameters and so the data is lost; arrays may be of different lengths.

Global data structures are those pieces of data, maybe in the form of files, databases or variables, that are existing through out the entire system. Every module has access to global data structures and may alter the contents thereof. The effect of this again may create some unusual combination of data which reveals an error in another module.

3.5: Integration testing

## Why Integration Testing is Required?

Why Integration Testing Is Required????



**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

3.5: Integration testing

## Types

- Modules are integrated by two ways.
  - Non-incremental Testing (Big Bang Testing)
  - Each Module is tested independently and at the end, all modules are combined to form a application
  - Incremental Module Testing.
    - There are two types by which incremental module testing is achieved.
      - Top down Approach
      - Bottom up Approach



Copyright © Capgemini 2015. All Rights Reserved 14

Discuss the advantages and disadvantages of the Non incremental testing and conclude that is more advantageous to do incremental testing

3.5: Integration testing

## Top Down Integration Testing

- Top Down Incremental Module Integration:
  - Firstly top 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.

```
graph TD; M1[M1] --- M2[M2]; M1 --- M3[M3]; M2 --- M5[M5]; M2 --- M6[M6]; M3 --- Stub[Stub];
```

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

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

3.5: Integration testing

## Top Down Integration Testing

- Integration approach can be done Depth first or Breadth-first.
- Top down testing
  - 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

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 16

3.5: 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

```
graph TD; Driver[Driver] --- A[A]; A --- C[C]; A --- B[B]; A --- D[D]
```

**Capgemini**  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

3.5: Integration testing

## Bottom Up Integration Testing

- Bottom-Up testing
  - 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

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

3.5: Integration testing

## Top Down vs Bottom Up Testing

| Top Down Testing  |  |
|---|--|
| Advantages  | Disadvantages  |
| Advantageous if major flaws occur toward the top of the program           | Stub modules must be produced  |
| Once the I/O functions are added, representation of test cases are easier | Stub Modules are often more complicated than they first appear to be.                      |
| Early skeletal Program allows demonstrations and boosts morale            | Before the I/O functions are added, representation of test cases in stubs can be difficult |
|   | Test conditions may be impossible, or very difficult, to create                            |
|   | Observation of test output is more difficult   |
|   | Allows one to think that design and testing can be overlapped                              |
|   | Induces one to defer completion of the testing of certain modules.                         |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 19

3.5: Integration testing

## Top Down vs Bottom Up Testing

| Bottom Up testing  |  |
|--|--|
| Advantages   | Disadvantages  |
| Advantageous if major flaws occur toward the bottom of the program | Driver Modules must be produced  |
| Test conditions are easier to create                               | The program as an entity does not exist until the last module is added |
| Observation of test results is easier                              |  |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

3.6. System testing

## Introduction

- Test the software in the real environment in which it is to operate. (hardware, people, information, etc.)
- Observe how the system performs in its target environment, for example in terms of speed, with volumes of data, many users, all making multiple requests.
- Test how secure the system is and how can the system recover if some fault is encountered in the middle of procession.
- System Testing, by definition, is impossible if the project has not produced a written set of measurable objectives for its product.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Discuss Airline Reservation Case Study and relate each test case with the System Tests that we are going to deal with in the next few slides and also the ones that were previously discussed

3.6: System testing

## Types of System Testing

- Types of System Testing
  - Functional testing
  - Regression testing
  - Performance
  - Volume
  - Stress
  - Security
- Usability
- Recovery
- Documentation
- Configuration
- Installation

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 22

3.6: System testing

## Functional Testing

- The main objective of functional testing is to verify that each function of the software application / system operates in accordance with the written requirement specifications.
- It is a black-box process
  - Is not concerned about the actual code.
  - Focus is on validating features.
- Uses external interfaces, including Application programming interfaces (APIs), Graphical user interfaces (GUIs) and Command line interfaces (CLIs).

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 23

3.6: System testing

## 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.
- Test suite contains -
  - Sample of tests that will exercise all software functions
  - Tests that focus on software functions that are likely to be affected by the change
  - Tests for software components that have been changed

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 24

3.6. System testing

## Performance Testing

- **Performance**
  - Performance is the behavior of the system w.r.t. goals for time, space, cost and reliability
- **Performance objectives:**
  - Throughput : The number of tasks completed per unit time. Indicates how much work has been done within an interval
  - Response time : The time elapsed during input arrival and output delivery
  - Utilization : The percentage of time a component (CPU, Channel, storage, file server) is busy

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

3.6 System testing

## Performance Testing

- The objective of performance testing is to devise test case that attempts to show that the program does not satisfy its performance objectives.
- To ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.
- To test response time and reliability by increased user traffic.
- To identify which components are responsible for performance degradation and what usage characteristics cause degradation to occur.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 26

3.6: System testing

## Volume Testing

- This testing is subjecting the program to heavy volumes of data. For e.g.
  - A compiler would be fed a large source program to compile
  - An operating systems job queue would be filled to full capacity
  - A file system would be fed with enough data to cause the program to switch from one volume to another.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

3.6: System testing

## Stress Testing

- Stress testing involves subjecting the program to heavy loads or stresses.
- The idea is to try to “break” the system.
- That is, we want to see what happens when the system is pushed beyond design limits.
- It is not same as volume testing.
- A heavy stress is a peak volume of data encounters over a short time.
- In Stress testing a considerable load is generated as quickly as possible in order to stress the application and analyze the maximum limit of concurrent users the application can support.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 28

3.6: System testing

## Stress Testing

- Stress tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Example :
  - Generate 5 interrupts when the average rate is 2 or 3
  - Increase input data rate
  - Test cases that require max. memory
- Stress Tests should answer the following questions
  - Does the system degrade gently or does the server shut down
  - Are appropriate messages displayed ? E.g. Server not available
  - Are transactions lost as capacity is exceeded
  - Are certain functions discontinued as capacity reaches the 80 or 90 percent level

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 29

3.6. System testing

## Security Testing

- Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.
- Security testing is the process of executing test cases that subvert the program's security checks.
- Example :
  - One tries to break the operating systems memory protection mechanisms
  - One tries to subvert the DBMS's data security mechanisms
  - The role of the developer is to make penetration cost more than the value of the information that will be obtained

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 30

Any computer based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities : hackers who attempt to penetrate systems for sport, disgruntled employees who attempt to penetrate for revenge, dishonest individuals who attempt to penetrate for illicit personal gain.

**Firewalls** – a filtering mechanism that is a combination of hardware and software that examines each incoming packet of information to ensure that it is coming from a legitimate source, blocking any data that are suspect.

**Authentication** – a verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.

**Encryption** – Protect sensitive data by modifying it in a way that makes it impossible to read by those with malicious intent.

**Authorization** – a filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes. (e.g. Userid , Passwords)

3.6: System testing

## Web Security Testing

- Web application security is a branch of Information Security that deals specifically with security of web applications.
- It provides a strategic approach in identifying, analyzing and building a secure web applications.
- It is performed by Web Application Security Assessment.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 31

### Why, Web Application Security?

1. Top breaches/fraud in recent times
2. Heartland Payment Systems - In January 2009 attackers were able to steal more than 130,000,000 credit card records
3. Virginia State Prescription Monitoring Program Records - Hackers stole 8.3 million records, erased the originals and created an encrypted backup of VPMP's database
4. Terrorists intercept US Drone unencrypted Video Feeds - Islamic terrorists have been able to hack into CIA state-of-the-art Predator drones with the help of just a 25.95 dollar off-the-shelf software, raising fears of remote control operated unmanned crafts being taken over and used against British and American targets.
5. Phishing attacks on banking sites – ICICI, SBI etc.,

3.6: System testing

## Security Testing Vs Functional Testing

- Functional testing checks for:
  - Invalid links (outgoing/internal/broken)
    - Is forward/backward link representing to the correct page?
  - Validations in each fields in the forms or web pages
    - Is Username filed accepting any special characters?
    - Is date field accepting the correct format specified?
  - HTML/CSS syntactical errors Etc.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 32

Functional testing alone is not a primary defense against the real world attacks.

3.6: System testing

## Security Testing Vs Functional Testing

- Security testing is all about:
  - With the help of a hyper links is it possible-
    - To access restricted resources like documents?
    - To insert any piece of code that could do the damage?
    - To upload any executable program?
  - Client side validations in each fields-
    - Properly validated fields - can they be bypassed- YES
  - HTML/CSS syntactical errors
    - Error pages can reveal sensitive information
    - Even a single Quote- ' - can do the damage

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 33

Functional testing alone is not a primary defense against the real world attacks.

3.6. System testing

## Localization Testing

- Localization translates the product UI and occasionally changes some settings to make it suitable for another region.
- The test effort during localization testing focuses on
  - Areas affected during localization, UI and content
  - Culture/locale-specific, language specific and region specific areas

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 34

The goal of globalization testing is to detect potential problems in application design that could inhibit globalization. It makes sure that the code can handle all international support without breaking the functionality that would cause either data loss or display problems.

Automated testing is not an effective solution for localization/globalization testing because the default mouse click position changes with the language and also the text in the recordings don't obviously match for all languages.

3.6. System testing

## Usability Testing

- Usability is
  - The effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment ISO 9241-11
  - Effective— Accomplishes user's goal
  - Efficient— Accomplishes the goal quickly
  - Satisfaction— User enjoys the experience
- Test Categories and objectives
  - Interactivity ( Pull down menus, buttons)
  - Layout
  - Readability
  - Aesthetics
  - Display characteristics
  - Time sensitivity
  - Personalization

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 35

Usability testing can be formal, informal or heuristic based on the needs and the availability.

There are many frameworks formulated for usability testing like NIST has formulated a CIF (Common Industry Format) for reporting the findings of the test.

Interactivity – Are interaction mechanisms( pull down menus ,pointers) easy to understand

Layout – Are navigation mechanisms, content and functions placed in a manner that allows the user to find them quickly?

Readability – Is text well written and understandable ? Are graphic representation easy to understand?

Aesthetics – Do layout, color, typeface and related characteristics lead to ease of use?

Do users feel comfortable with the look and feel.

Display Characteristics – optimal use of screen size and resolution.

Time sensitivity – Can important features, functions and content be used or acquired in a timely manner

Personalization – Does the web application tailor itself to the specific needs of different user categories or individual users?

3.6. System testing

## Usability Testing

- Using specialized Test Labs a rigorous testing process is conducted to get quantitative and qualitative data on the effectiveness of user interfaces
- Representative or actual users are asked to perform several key tasks under close observation, both by live observers and through video recording
- During and at the end of the session, users evaluate the product based on their experiences

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 37

Trainer notes

3.6: System testing

## Recovery Testing

- A system test that forces the software to fail in variety of ways, checks performed
  - recovery is automatic ( performed by the system itself)
  - reinitialization
  - check pointing mechanisms
  - data recovery
  - restarts are evaluated for correctness
- This test confirms that the program recovers from expected or unexpected events. Events can include shortage of disk space, unexpected loss of communication

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 38

It Confirms that the program recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.

3.6 System testing

## Documentation Testing

- This testing is done to ensure the validity and usability of the documentation
- This includes user Manuals, Help Screens, Installation and Release Notes
- Purpose is to find out whether documentation matches the product and vice versa
- Well-tested manual helps to train users and support staff faster

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 39

Generally, Technical Writers work to tight schedules, which often does not include documentation testing because there is no time. Besides, no one wants to take the risk of causing a rewrite or correcting product design and not shipping on schedule.

Bad documentation has a ripple effect on the number of users it impacts such as Product Development, Training, and Customer Support.

3.6. System testing

## Configuration Testing

- Attempts to uncover errors that are specific to a particular client or server environment
- Create a cross reference matrix defining all probable operating systems, browsers, hardware platforms and communication protocols
- Test to uncover errors associated with each possible configuration

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 40

### Configuration Test

Analyze system behaviour:

- in various hardware and software configurations specified in the requirements
- sometimes systems are built in various configurations for different users
- for instance, a minimal system may serve a single user, other configurations for additional users.

3.6: System testing

## Installation Testing

- Installer is the first contact a user has with a new software!!!
- Installation testing is required to ensure:
  - Application is getting installed properly
  - New program that is installed is working as desired
  - Old programs are not hampered
  - System stability is maintained
  - System integrity is not compromised

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 41

3.7: User Acceptance Testing

## Introduction

- 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
- Not a responsibility of the Developing Organization
- To test whether or not the right system has been created
- Usually carried out by the end user
- Two types are :
  - ALPHA TESTING: Generally in the presence of the developer at the developers site
  - BETA TESTING: Done at the customers site with no developer in site

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 42

3.8: Exploratory Testing

## Introduction

- Also known as “Random” testing or “Ad-hoc” testing
- Exploratory testing is simultaneous learning, test design, and test execution. (...James Bach)
- A methodical approach-style is desirable

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 43

Among the hardest things to explain is something that everyone already knows. We all know how to listen, how to read, how to think, and how to tell anecdotes about the events in our lives. As adults, we do these things everyday. Yet the *level* of any of these skills, possessed by the average person, may not be adequate for certain special situations. Psychotherapists must be *expert* listeners and lawyers *expert* readers; research scientists must scour their thinking for errors and journalists report stories that transcend parlor anecdote. So it is with exploratory testing (ET): simultaneous learning, test design and test execution.

3.8: Exploratory Testing

## Tips

- Test design Crafting
- Careful Observation
- Critical thinking
- Diverse Ideas
- Pooling resources (knowledge, learnings)

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 44

1. **Test Design:** An exploratory Test Engineer is first and foremost a test designer. Anyone can design a test accidentally, the excellent exploratory tester is able to craft tests that systematically explore the product. That requires skills such as the ability to analyze a product, evaluate risk, use tools, and think critically, among others.
2. **Careful Observation:** Excellent exploratory testers are more careful observers than novices, or for that matter, experienced scripted testers. The scripted tester need only observe what the script tells him to observe. The exploratory tester must watch for *anything* unusual or mysterious. Exploratory testers must also be careful to distinguish observation from inference, even under pressure, lest they allow preconceived assumptions to blind them to important tests or product behavior.
3. **Critical Thinking:** Excellent exploratory testers are able to review and explain their logic, looking for errors in their own thinking. This is especially important when reporting the status of a session of exploratory tests, or investigating a defect.
4. **Diverse Ideas:** Excellent exploratory testers produce more and better ideas than novices. They may make use of heuristics to accomplish this. Heuristics are mental devices such as guidelines, generic checklists, mnemonics, or rules of thumb.
5. **Rich Resources:** Excellent exploratory testers build a deep inventory of tools, information sources, test data, and friends to draw upon. While testing, they remain alert for opportunities to apply those resources to the testing at hand.

3.9: Test Strategy

## What Is a Test Strategy?

- It provides a road map that describes the steps to be conducted as part of testing
- When these steps are planned then how much effort, time and resources will be required are undertaken
- It must incorporate test planning, test case design, test execution and resultant data collection and evaluation

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 45

### Test Strategy – Generic characteristics:

- To perform effective testing a software should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise , progress must be measurable and problems must surface as early as possible.

## Summary

- In this lesson, you have learnt:
  - Verification refers to a set of activities which ensures that software correctly implements a specific function.
  - Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
  - Different testing phases are
    - Unit testing
    - Integration testing
    - System testing
    - Acceptance testing
  - Exploratory testing is simultaneous learning, test design and test execution.



## Review Question

- Question 1: \_\_\_\_\_ is a Quality improvement process
  
- Question 2: Volume tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
  - True/False
  
- Question 3: Acceptance testing is not a responsibility of the Developing Organization
  - True/False



## Review Question: Match the Following

1. Beta testing
2. Response time
3. Aesthetics

- A. Volume testing
- B. Exploratory testing
- C. Acceptance testing
- D. Documentation testing
- E. Performance testing
- F. Usability testing



# **Testing Concepts for V&V Automation Testing**

Lesson 4: Test Case  
Execution/Testing Metrics

## Lesson Objectives

- Test Case execution
- The objective is to understand various metrics that evaluate the effectiveness of the testing process
- To learn best practices for test case maintenance



Copyright © Capgemini 2015. All Rights Reserved. 2

4.1: Test Case Execution

## Pre-execution activities

- Setting up the Environment
  - Similar to production environment
  - Hardware (e.g. Hard Disk, RAM, Processor)
  - Software (e.g. IE, MS office)
  - Access to Applications
- Setting up data for Execution
  - Any format (e.g. xml test data, system test data, SQL test data)
  - Create fresh set of your own test data
  - Use existing sample test data
  - Verify, if the test data is not corrupted
  - Ideal test data - all the application errors get identified with minimum size of data set

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

## 4.1: Test Case Execution

## Pre-execution activities contd...

- Test data to ensure complete test coverage
- Design test data considering following categories:
  - No data
    - Relevant error messages are generated
  - Valid data set
    - Functioning as per requirements
  - Invalid data set
    - Behavior for negative values
  - Boundary Condition data set
    - Identify application boundary cases
  - Data set for Performance, Load and Stress Testing
    - This data set should be large in volume



Copyright © Capgemini 2015. All Rights Reserved 4

4.1: Test Case Execution

## Types of Test Environment

- Unit Test Environment
- Assembly/Integration Test Environment
- System/Functional/QA Test Environment
- User Acceptance Test Environment
- Production Environment



Copyright © Capgemini 2015. All Rights Reserved 5

4.1: Test Case Execution

## Before starting Execution

- Validate the Test Bed
  - Environment
    - Hardware (e.g. Hard Disk, RAM, Processor)
    - Software (e.g. IE, MS office)
  - Access
    - Access to the Application
    - Availability of Interfaces (e.g. Printer)
    - Availability of created Test Data
  - Application
    - High level testing on the application to verify if the basic functionality is working
    - There are no show-stoppers
    - Referred to as Smoke/Sanity/QA Build Acceptance testing

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 6

4.1: Test Case Execution

## Entry Criteria for Functional Testing

| Functional/System Testing Entry Criteria   |
|--|
| <ul style="list-style-type: none"><li>Integration Testing is complete and sign-off is received by Project team</li></ul>   |
| <ul style="list-style-type: none"><li>Integration test results are provided to the QA team within the Integration Execution &amp; Signoff artefact.</li></ul>          |
| <ul style="list-style-type: none"><li>Development team provides a demonstration of application changes prior to promotion to QA Environment</li></ul>                  |
| <ul style="list-style-type: none"><li>Code is delivered and successfully promoted to the Functional/System Test Environment as described in Master Test Plan</li></ul> |
| <ul style="list-style-type: none"><li>Functional/System Test planning is detailed, reviewed and approved within the Master Test Plan</li></ul>                         |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 7

4.1: Test Case Execution

## Entry Criteria for Functional Testing Contd...

| Functional/System Testing Entry Criteria  |
|---|
| ➤ Smoke /Shake down test has been completed to ensure test environment is stable for testing.                     |
| ➤ Functional/System Test planning is detailed, reviewed and approved within the Master Test Plan                  |
| ➤ Test cases created are traceable back to SRS, and any approved change requests, using HPQC                      |
| ➤ Functional/System Test Cases are created, reviewed and approved within the RBC Enterprise approved tool (HP QC) |
| ➤ Test data is ready for Functional/System Testing  |

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 8

4.1: Test Case Execution

## Execution

- Run Tests
  - Run test on the identified Test Bed
  - Precondition
  - Use the relevant test data
- Note the Result
  - Objective of test case
  - Action performed
  - Expected outcome
  - Actual outcome
  - Pass/Fail (according to pass/fail criteria)

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

4.1: Test Case Execution

## Execution

- Compare the Input and Output
  - Validate the data (e.g. complex scenarios, data from multiple interfaces)
- Record the Execution
  - Test data information (e.g. type of client, account type)
  - Screenshots of the actions performed and results
  - Video recording (HP QC Add-in)

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 10

4.1: Test Case Execution  
**Execution contd...**

- Report deviation
  - Log Defect for Failed Test Cases
  - Defect logging
    - Project
    - Summary
    - Description
    - Status
    - Detected By
    - Assigned To
    - Environment (OS, Release, Build, Server)
    - Severity
    - Priority
    - Steps to recreate and Screenshots



Copyright © Capgemini 2015. All Rights Reserved 11

4.1: Test Case Execution

## Exit Criteria for Functional Testing

- **Functional/System Testing Exit Criteria**

- All high and medium risk tests identified in the detailed test plan are executed, including interface testing
- All planned testing is complete and documented
- Functional/System test execution results are captured
- All known defects have been entered into the defect tracking tool
- There are no known severity one or severity two defects
- Action plans have been created for outstanding severity three and four defects

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 12

4.1: Test Case Execution

## Exit Criteria for Functional Testing Contd...

### Functional/System Testing Exit Criteria

- Appropriate signoffs are obtained
- Location of test cases, automated test scripts, defects and Functional/System Execution & Signoff artefact are detailed within the SCM plan.
- Any known deviations from the BRD and SRS are documented and approved



Copyright © Capgemini 2015. All Rights Reserved 13

4.2: Testing metrics

## Need and definition of Metrics

- Metrics - definition
  - A metric is the measurement of a particular characteristic of a program's performance or efficiency.
  - A quantitative measure of the degree to which a system, component or process possesses a given attribute.

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Process measurement and analysis, and utilization of quantitative methods for quality management are the two key process activities at Level 4 Maturity of CMM.

While the need for metrics has been recognized, implementation of structured measurement programs is lagging, especially in the software testing area.

Efficient test process measurement is essential for managing and evaluating the effectiveness of a test process.

Test metrics are an important indicator of the effectiveness of a software testing process.

There are various metrics that evaluate the effectiveness of the testing process and also the other metrics that are affected due to the testing process.

In general, we measure following things

1. Functionality
2. Schedule
3. Cost

4.2: Testing metrics

## Need and definition of Metrics

- Why Measure
  - Tracking Projects against plan
  - Take timely corrective actions
  - Getting early warnings
  - Basis for setting benchmarks
  - Basis for driving process improvements
  - Tracking process performance against business

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

- Test Metrics data collection helps predict the long term direction and scope for an organization
- Provides a basis for estimation and facilitates planning for closure of the performance gap
- Provides a means for control/status reporting
- Identify critical processes that will be monitored statistically
- Identifies risk areas that require more testing
- Provides meters to flag actions for faster and more informed decision making
- Helps in identifying potential problems and areas of improvement
- Provide an objective measure of the effectiveness and efficiency of testing

4.2: Testing metrics

## Metrics for Testing

- Defect Density
  - Total Defect density = (Total number of defects including both impact and non-impact, found in all the phases + Post delivery defects)/Size
- Average Defect Age
  - Average Defect age = (Sum of ((Defect detection phase number – defect injection phase number) \* No of defects detected in the defect detection phase))/(Total Number of defects till date)
- Defect Removal Efficiency
  - DRE = 100 \* No. of pre-delivery defects / Total No. of Defects

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

**Defect Density is the no. of defects found in a unit.**

Defects are measured as found in

1. Reviews
2. Testing
3. Acceptance
4. Warranty

The defect that are hampering the functionality are ‘impact’ defects. Defect those are not affecting functionality like look n feel errors, displacement errors are non-impact errors.

The defect impacting the functionality are direct. Impact defect whereas the commenting standard. Defect found in code review is non-impact.

E.g. If Total number of direct impact defects found in all the phases = 20, Post delivery defects = 10, Size = 100, Direct Impact Defect density =  $20+15/100 = 0.30$

We should try n minimize it. We should minimize impact & non-impact errors.

**Average Defect Age**

The Average defect age tells us for how long the defect was in the system after it was injected.

For E.g. If Defect detection phase number= 4, defect injection phase number = 2, No of

defects detected in the defect detection phase = 10, Total Number of defects till date = 40

Defect Age =  $4-2*20/40 = 1$  ----- This is the defect age. Should be kept minimum.

Difference should be kept minimum.

Average Defect age is the sum of defects detected at all the stages.

**Defect Removal Efficiency**

For E.g. If No. of pre-delivery defects = 5, Total No. of Defects = 20,

DRE =  $100 * 5/20 = 100*0.25=25$

This has to be 100%. No of pre-delivery defects should be greater than total no. of defects. So

the errors should get find before delivery.

Low Defect Removal Efficiency means – More defects left undetected before delivery,

Reviews and Testing failed to detect them.

4.2: Testing metrics

## Metrics for Testing

- **Review Effectiveness**
  - $\text{Review Effectiveness} = 100 * \text{Total no. of defects found in review} / \text{Total no. of defects}$
- **Cost of finding a defect in review(CFDR)**
  - $\text{Cost of finding a defect in reviews} = (\text{Total efforts spent on reviews} / \text{No. of defects found in reviews})$
- **Cost of finding a defect in testing(CFDT)**
  - $\text{Cost of finding a defect in testing} = (\text{Total efforts spent on testing} / \text{defects found in testing})$

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

### **Review Effectiveness :**

The review effectiveness tells us how effective is review process. If all the defects are found during the review then the effectiveness % will be 100.

E.g. If Total no. of defects found in review = 20, Total no. of defects=40

$\text{RE} = 100 * 20 / 40 = 50$ . So defects should get find at review stage only to achieve 100 % effectiveness.

Low Review Effectiveness means : More defects detected in testing. Reviews failed to detect early

### **Cost of finding defect in review**

This metric tells us the effort spent in finding a defect in reviews. Cost of reviews include all the efforts spent in review briefing, defect recording etc. This includes reviewer, recorder and creators time if creator is attending the review his/her time also should be recorded.

For E.g. If Total efforts spent on reviews = 40 hrs, No. of defects found in reviews = 20, CFDR=  $40 / 20 = 2$  Hrs

### **Cost of finding defect in testing**

This metric computes the Cost of finding a defect in testing . Total time spent on testing includes time to create and review, run the test cases and recording the defects. This should not include the time spent in fixing the defects.

**CFDT = Total efforts spent on testing / defects found in testing = 60/30=2 hrs**

4.2: Testing metrics

## Metrics for Testing

- Components of CoQ – Prevention Cost, Appraisal Cost, Failure Cost
- Prevention Cost: (Green Money)
  - Cost of time spent in DP meetings
  - Cost of time spent by DPR/PM/TL on analysis of defect entries/discussions with team members
  - Cost of time spent by the team in implementing the preventive actions identified from project start date to till date
- Appraisal Cost: (Blue Money)
  - Cost of time spent on review and testing activities from the project start date to till date
- Failure Cost: (Red Money)
  - Failure costs include internal and external failure costs
  - Cost of time taken to fix the pre and post delivery defects
  - Expenses incurred in rework – Customer does not pay for this

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 19

**Prevention** - Money required preventing errors and to do the job right the first time is considered prevention cost. This category includes money spent on establishing methods and procedures, training workers and planning for quality. Prevention money is all spent before the product is actually built.

**Appraisal** – Appraisal costs cover money spent to review completed products against requirements. Appraisal includes the cost of inspections, testing and reviews. This money is spent after the product or subcomponents are built but before it is shipped to the user.

**Failure** – Failure costs are all costs associated with defective products. Some failure costs involve repairing products to make them meet requirements. Others are costs generated by failures, such as the cost of operating faulty products, damage incurred by using them and the costs incurred because the product is not available. The user or customer of the organization may also experience failure costs.

4.2: Testing metrics

## Metrics for Testing

- Money spent beyond what it would cost to build a product right first time
- **Cost of Quality**
  - % Cost of Quality =  $(\text{Total efforts spent on Prevention} + \text{Total efforts spent on Appraisal} + \text{Total efforts spent on failure or rework}) * 100 / (\text{Total efforts spent on project})$
  - Failure cost = Efforts spent on fixing or reworking the pre-delivery defects +  $(3 * \text{efforts spent on fixing or reworking the post-delivery defects})$

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

**Cost of Quality** - Cost of Quality consists of Prevention cost, Appraisal cost & Failure (or Rework) cost. Here, cost is the efforts measured in terms of person days.

**Prevention cost consists of efforts spent on preventing defects such as:**

1. Time spent in various Defect Prevention meetings
2. Time spent by Defect Prevention Reviewer/Project Leader on analysis of defect entries/discussions with team members/SQA
3. Time spent by the team in implementing the preventive actions identified from project start date to till date.

For E.g. If Total efforts spent on Prevention = 20, Total efforts spent on Appraisal = 30, Total efforts spent on failure or rework = 40, Total efforts spent on project = 140

**Cost of quality** =  $(20+30+40)*100/140= 64$ . This has to decrease. Total efforts of all the activities should match with efforts spent on time.

Failure cost = Efforts spent on fixing or reworking the pre-delivery defects +  $(3 * \text{efforts spent on fixing or reworking the post-delivery defects})$

(As the impact of post delivery defects will be high, weightage of "3" has been attached to it)

For E.g. Efforts spent on fixing or reworking the pre-delivery defects = 40, efforts spent on fixing or reworking the post-delivery defects = 20 **Failure cost** =  $40+(3*20)= 40+60= 100$ .

We need to keep it minimum. So, efforts for post-delivery defects should be 0 to minimize failure cost.

4.2: Testing metrics

## Metrics for Testing

- Test Case Effectiveness
  - Test Case Effectiveness = # of defects detected using the test cases \* 100/ total # of defects detected in testing
  - This metrics defines the effectives of the test cases which is measured in terms of the number of defects found in testing with using the test cases
  - Source of Data
- Defect data and number of test cases from test execution report
  - P.S.: - These metrics are mainly applicable to V&V projects

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

**Test Case Effectiveness (TCE)** : This metrics defines the effectives of the test cases which is measured in terms of the number of defects found in testing without using the test cases.

**Test Case Effectiveness** = # of defects detected using the test cases \* 100/  
total #  
Of defects detected in testing

E.g. If # of defects detected using the test cases = 30, total # of defects detected in testing = 50

**Test Case Effectiveness = 30\*100/50 = 60**

Effectiveness needs to be high. # of defects detected using the test cases should match total # of defects detected in testing to have effectiveness.

4.2: Testing metrics

## Metrics for Testing

- Test Case Adequacy
  - Test Case Adequacy = No. of actual Test cases \* 100 / No. of test cases estimated
  - This metrics defines the number of actual test cases created vs. the estimated test cases at the end of the test case preparation phase
  - The estimated No. of the test cases are based baseline figures and then added to test plan
  - Number of Actual Test cases is also derived from project plan
- P.S.: - These metrics are mainly applicable to V&V projects

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

### Test Case Adequacy

This metrics defines the number of actual test cases created vs. the estimated test cases at the end of the test case preparation phase. The estimation of the planned test cases are based upon the baseline figures.

#### For E.g.

If No. of actual Test cases = 30, No. of test cases estimated = 40  
Test Case Adequacy =  $30 \times 100 / 40 = 75$

This has to be 100%. For which, No. of actual Test cases should match No. of test cases estimated. There should not be vast difference between them to achieve higher adequacy.

4.2: Testing metrics

## Metrics for Testing

- Defect Detection Index
  - Defect Detection Index = # of defects detected in each phase / total # of defects planned to be detected in each phase
  - This is a measure of actual vs. planned defects at the end of each phase
    - Defect data from execution report
- P.S.: - These metrics are mainly applicable to V&V projects

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

**Defect Detection Index (DDI)** - This is a measure of actual vs. planned defects at the end of each phase.

For E.g. If # of defects detected in each phase = 20, total # of defects planned to be detected in each phase = 100

Defect Detection Index =  $20/100 = 0.50$

4.2: Testing metrics

## Metrics for Testing

- Test Coverage: The following are the test coverage metrics:
- Test Design:
  - # Of Requirements or # Of Use Cases covered / # Of Requirements or # Of Use Cases Planned
- Test Execution:
  - # Of Test scripts or Test cases executed/# Of Test scripts or Test cases Planned
- Test Automation:
  - # Of Test cases automated/# Of Test cases

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

4.2: Testing metrics

## Metrics for Testing

- Test Effectiveness
  - # Of Test Cases failed (found defects)/# Of Test Cases executed
- Delivered Defect Rate (Per 1000 Person Hours)
  - (# Of Defects \* 1000)/ Actual Effort
- Defect Injection Rate (No of Defects / 100 Person Hours)
  - No of Defects[phase wise] \* 100/ Actual Effort[phase wise]
- Defect Removal efficiency
  - (# of Defects found internally / Total # Of(internal + external) Defects found) \* 100

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

### Test Effectiveness

# Of Test Cases failed (found defects)/# Of Test Cases executed

This metric indicates the effectiveness of the Test Cases in finding the defects in the product

### Delivered Defect Rate (Per 1000 Person Hours)

(# Of Defects \* 1000)/ Actual Effort

The purpose of this parameter is to measure the defect slippage to our customer vis-à-vis total effort. This parameter will be used to predict the residual defects in the delivered product with our current capability.

### Defect Injection Rate (No of Defects / 100 Person Hours)

No of Defects [phase wise] \* 100/ Actual Effort [phase wise]

This is used to detect the defects injected during STLC Phases.

### Defect Removal efficiency

(# of Defects found internally / Total # Of(internal + external) Defects found) \* 100

It indicates the number of defects leaked after several levels of review and these defects are slipped to the customer.

This is same as review effectiveness. Need to discuss before removing.

4.2: Testing metrics

## Types of Metrics

- There are several types of metrics
  - Project Metrics
  - Process Metrics
  - Productivity Metrics
  - Closure Metrics

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 26

4.2: Testing metrics

## Metrics for Testing – Project Metrics

- The following are the Project Metrics
  - Test Coverage
  - Defect Density
  - Defect arrival rate

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 27

**Defect arrival rate:**

# Of Defects \* 100 / # of Test Cases planned for Execution

This metric indicates the quality of the application/product under test.  
Lower the value of this parameter is better.

4.2: Testing metrics

## Metrics for Testing – Process Metrics

- The following are the Process Metrics
  - Test Effectiveness
  - Effort Variance
  - Schedule Variance
  - Cost of Quality
  - OTD
  - Delivered Defect Rate
  - Defect Slippage or Test escape

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 28

### Effort Variance

(Overall and Variance at each milestone)

$((Actual\ effort - Planned\ effort) / Planned\ effort) * 100$

The purpose of this parameter is to check the accuracy of the Effort estimation process to improve the estimation process.

### Schedule variance

$(Actual\ end\ date - Planned\ end\ date) / (Planned\ end\ date - Plan\ start\ date + 1) * 100$

It depicts the  $\pm$  buffer that can be used for optimum use of resource deployment And monitor the dates committed to the client and helps in better planning of future tasks.

### OTD

# of deliveries on schedule / total # of deliverables made

This parameter is to measure the timely delivery of the Deliverables.

### Defect slippage or Test escape

$(Total\ #\ Of\ External\ Defects / Total\ #\ Of\ Defects\ detected\ (Internal+External)) * 100$

This measure helps us to know how effectively we are detecting the defects Various stages of internal testing

4.2: Testing metrics

## Metrics for Testing – Process Metrics

- The following are the Process Metrics
  - Defect Injection Rate
  - Rejection Index
  - Resource Utilization
  - Review Effectiveness
  - Test Case Design Rework Index
  - Defect Removal Efficiency

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 29

### Rejection index

# Of Defects rejected/# Of Defects raised

The purpose of this parameter is to measure the Quality of the defects raised

### Resource Utilization

Actual effort utilized in the month for project activities /Total available Effort in the month

### Review Effectiveness

(No of Internal Review Defects / [No of Internal Defects+No of External Defects])\*100

The purpose of this parameter is to measure how effective are our reviews in capturing all the phase injected defects.

### Test Case Design Rework Index

(Test Cases with Review Comments for rework/Total Test Cases)\*100

4.2: Testing metrics

## Metrics for Testing – Productivity Metrics

- The following are the Productivity Metrics
  - Test case design productivity
  - Test case execution productivity

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 30

### Test case design productivity

# Of Test cases (scripts) designed/ Total Test case design effort in hours  
This metric indicates the productivity of the team in designing the test cases.

### Test case execution productivity

# Of Test cases executed/ Total Test case executed effort in hours  
Effort shall include the set up time, execution time. This metric indicates the productivity of the team in executing the test cases.

4.2: Testing metrics

## Metrics for Testing – Closure Metrics

- The following are the Closure Metrics Effort distribution
  - Test Design Review Effort
  - Test Design Rework effort
  - KM Effort

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 31

**Effort Distribution**

(Effort spent on a STLC Phase / Total STLC effort) \* 100

This would indicate the effort distribution across STLC phases. Given the type of an engagement and application, we can re-use this in estimation & planning for future projects

**Test Design Review effort**

(Effort spent on Test Case design reviews / Total effort for spent on Test Case design) \*

100

This can be used with other process metrics like "Review Effectiveness", "Defect removal Efficiency", "Defect Injection Ratio" to plan for an adequate review effort for future projects.

**Test Design Rework effort**

(Effort spent on Test Case design review rework / Total effort for spent on Test Case design) \* 100

This can be used with other process metrics like "Effort Variance", "Schedule Variance" to plan for an adequate rework effort for future projects.

**KM Effort**

(Total Effort spent on preparation of the KM artifacts / Total actual effort for the project)

\* 100

This indicates the effort spent on KM. This along with other process and product metrics can be used to plan for KM activities for future projects.

4.3: Test case maintenance

## Best practices for test case maintenance

- Have Approved Test Case template in place
- Identify the location of Test Cases storage with good access control
- Have Test Case Review & Approval SOP in place
- Appropriate Training for Testers for Test Case Authoring / Reviews / Executions / Maintenance
- Test Cases attributes can be discussed and agreed upon:
  - Should it contain Navigational OR click-by-click Test Steps
  - Should it contain Test Data set up steps within Test Case or it should be done separately
  - Test Case modification protocol
  - Reusable components development

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 33

It's really important to think about how you structure and divide your test cases to make them as reusable as possible.

For example, you need to write test cases to test to flow say

Login->View cosmetic product->Select cosmetic product->Checkout.

You should write a short test case for each function, rather than writing a huge test case that tests the entire flow. In this way, you can reuse the same test cases even if the flow changes, merely re-tying them.

4.3: Test case maintenance

## Best practices for test case maintenance

- Audit Trail for every update in Test Case
- Good management of Impact assessments with every update in requirement(s) w.r.t Test Cases coverage
- Better management of Trace Matrix with “every” update
- Maintenance of record of Executed Test Cases and Defects for reference of updates
- It is advisable to store test cases in version control tool so that any subsequent changes can be tracked easily
- Use test case creation and maintenance tools. One such tool is Quality Center from HP

 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 34

## Summary

- In this lesson, you have learnt:
  - Various testing metrics like
    - Defect Density
    - Average Defect Age
    - Defect Removal Efficiency
    - Review Effectiveness
    - Cost of finding a defect in review
    - Cost of finding a defect in testing
    - Cost of Quality
    - Test Case Effectiveness
    - Test Case Adequacy
    - Defect Detection Index



## Review - Questions

- Question 1: The defect impacting the functionality are \_\_\_\_\_.(Indirect/Direct/Standard)
- Question 2: CFDR metric tells us the effort spent in finding a defect in testing
  - Option: True / False
- Question 3: Metrics are used to evaluate the effectiveness of the testing process
  - Option: True / False



## Review - Questions

- Question 4: Which of the following are Test Environments
  - Unit Test Environment
  - QA Test Environment
  - Simple Test Environment
  - Product Environment



## Review – Match the Following

1. Minimum

2. Maximum

A. Average Defect Age

B. Defect Removal Efficiency

C. Cost of finding a defect  
in testing

D. Review Effectiveness

E. Cost of Quality

F. Test Case Adequacy



# Testing Concepts for V&V Automation Testing

Lab Book

## Document Revision History

---

| Date       | Revision No. | Author         | Summary of Changes                          |
|------------|--------------|----------------|---|
| 12/8/09    | 1            | Priya Rane     | Revamp                                      |
| 06/09/2011 | 2            | Hema G         | Revamp                                      |
| 19/06/2013 | 3            | Selvalakshmi P | Revamp                                      |
| 15/04/2015 | 4            | Dayanand Patil | Revamp                                      |
| 23/12/2015 | 5            | Shilpa Bhosle  | Customized for V&V – Automation Testing LoT |

## Table of Contents

---

|   |    |
|---|----|
| <i>Document Revision History</i> .....  | 2  |
| <i>Table of Contents</i> .....  | 3  |
| <i>Lab 1. Software Testing Basics</i> .....   | 4  |
| 1.1 Validate Date field.....  | 4  |
| 1.2 Validate Command Line utility.....  | 4  |
| 1.3 Validate Phone Number field.....  | 4  |
| 1.4 Validate Password Field .....   | 4  |
| <i>Lab 2. Creating Test Cases</i> .....   | 5  |
| 2.1 Case Study.....   | 5  |
| 2.2 Rules: Product Enquiry Form.....  | 5  |
| <i>Lab 3. Creating Test Cases</i> .....   | 8  |
| 3.1: Software Testing Case Study. 'ONLINE CONFERENCE ROOM BOOKING' on .....                   | 8  |
| Intranet .....  | 8  |
| Booking Instructions.....   | 8  |
| Making a new booking: .....   | 8  |
| Viewing / Cancellation of Bookings:.....  | 9  |
| <u>Invoke <a href="https://intranet.confBook.com">https://intranet.confBook.com</a></u> ..... | 9  |
| Intranet home →Employee Corner →Conference Room Booking .....                                 | 11 |
| Conference Room Booking → New Booking .....   | 11 |
| HOME →Conference Room Booking → New Booking .....   | 12 |
| If booking is successful, the following screen is displayed .....                             | 13 |
| HOME →Conference Room Booking → New Booking.....  | 13 |
| The booking status is also shown under My Bookings as shown below:- .....                     | 13 |
| HOME →Conference Room Booking → Booking Status .....  | 13 |
| Stretched Assignments .....   | 14 |
| 4.1 Customer Complaint Form.....  | 14 |
| 4.2 Instructions.....   | 14 |
| 4.3 Rules.....  | 14 |

## Lab 1. Software Testing Basics

---

|              |   |
|--------------|---|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>Understand the process of creating test cases.</li> <li>Learn to apply basic techniques for writing test cases.</li> <li>Learn to prepare finite test cases</li> </ul> |
| <b>Time</b>  | 90 Minutes  |

### 1.1 Validate Date field

Validate Date Field using Black Box Testing Techniques. Format of Date field is dd/mm/yyyy

### 1.2 Validate Command Line utility

Validate Command Line utility - 'MAX'.

The utility displays the maximum of the 2 specified Integers. Please note down any assumptions that you make.

E.g. MAX 2 3

Steps to run Max command line utility

1. create a folder demo on E drive
2. Copy max.exe file in to demo folder
3. Click on start > run. Type **cmd**
4. Type "E:"
5. Type " cd demo"

Use following command to run max utility

```
E:\demo> max 25 34
E:\demo> max 25 b
E:\demo> max a 34
E:\demo> max 25.45 34.67
```

### 1.3 Validate Phone Number field.

Format of the number is

Country Code (10 to 999) City Code (10 to 99999) Phone Number (1000000 to 99999999)

### 1.4 Validate Password Field

Write the test case for password field. Password should be the combination of Alphabets, numeric values and special characters. It should contain one upper case letter, at least one digit and at least one special character. The length of the password should be of minimum 8 characters.

## Lab 2. Creating Test Cases

---

|              |   |
|--------------|---|
| <b>Goals</b> | <ul style="list-style-type: none"> <li>• Read “Product Enquiry Form Instructions” before starting the assignment.</li> <li>• Develop creative test cases for the New Bike Enquiry Form</li> </ul> |
| <b>Time</b>  | 120 minutes   |

**Note:** In this lab use test case design techniques for designing creative test cases based on the requirements given below.

Participant are required to write at least 5 positive test cases, 10 Negative test cases, 5 Basic tests cases. This is minimum expectation.

### 2.1 Case Study

A customer can enquire about a new vehicle models by filling the “Product Enquiry Form” by visiting dealer’s website. This is useful for those customers who wish to upgrade their existing vehicle to a new one or wants to purchase a new vehicle. This website also facilitates a customer to book for a test ride through product enquiry form.

### 2.2 Rules: Product Enquiry Form

1. The fields marked as \* are mandatory fields on the form
2. Customer Name(firstname & lastname) should accept only characters
3. The mobile number should accept only numbers and it should contain only 10 digits
4. The telephone number field should accept only numbers and it should contain only 10 digits
5. The Email field should accept only valid email address e.g. **Dayanand.patil@domain.com**
6. The Vehicle Model should accept only following values
  - a. Pleasure
  - b. Karizma
  - c. Impulse
  - d. Splendor
  - e. Splendor+
7. Select the state from the state drop down box. This will automatically populate the district based on the selected state in a new drop down box. Refer the below given table for the valid values of states and district combination.

| State | District |
|-------|----------|
|       | Sangli   |

|             |                                       |
|-------------|---------------------------------------|
| Maharashtra | Kolhapur<br>Pune                      |
| Tamilnadu   | Chennai<br>Coimbatore<br>Kancheepuram |
| Gujarat     | Anand<br>Dohad<br>Rajkot              |

8. Select dealer's state from the Dealer State drop down box. This will automatically populate the Dealer drop down box. Refer the below given table for the valid values of Dealer's States and Dealers combination.

| Dealer State | Dealer                                |
|--------------|---------------------------------------|
| Maharashtra  | M/s Ghatge Patil auto and firm,Sangli |
|              | M/s Unique Automobiles(I),Kolhapur    |
|              | M/s Lakshya Motors, Pune              |
| Tamilnadu    | M/s Mohan Automobiles, Chennai        |
|              | M/s Suguna Automobiles, Coimbatore    |
|              | M/s Anand Automobiles,Kancheepuram    |
| Gujarat      | M/s Prakash Motors,Anand              |
|              | M/s Shree Laxminarayan Motors,Dohod   |
|              | M/s Aan Automobiles,Rajkot            |

9. The Brief about enquire field should accept maximum 100 characters max.

## Product Enquiry & Book A Test Ride

|                                 |                 |                                  |
|---------------------------------|-----------------|----------------------------------|
| Customer Name *                 | FirstName       | LastName                         |
| Moblie No. *                    |                 | Tel.                             |
| Email                           |                 |                                  |
| Vehicle Model *                 | Select          | <input type="button" value="▼"/> |
| State *                         | Select State    | <input type="button" value="▼"/> |
| District *                      | Select District | <input type="button" value="▼"/> |
| City *                          |                 |                                  |
| Existing Vechicle *             | Select Vehicle  | <input type="button" value="▼"/> |
| When would you like a Test Ride |                 |                                  |
| Dealer State *                  | Select State    | <input type="button" value="▼"/> |
| Dealer *                        | Select Town     | <input type="button" value="▼"/> |
| Brief About Enquiry             |                 |                                  |

10. The website will validate the data for its correctness.
11. The customer need to fill the required details and should click on the Submit button to submit the enquiry with the website.

## Lab 3. Creating Test Cases

---

|              |   |
|--------------|---|
| <b>Goals</b> | <ul style="list-style-type: none"><li>• Read 'BOOKING INSTRUCTIONS' before starting the assignment.</li><li>• Understand the application and develop creative test cases.</li></ul> |
| <b>Time</b>  | 120 minutes   |

### 3.1: Software Testing Case Study. 'ONLINE CONFERENCE ROOM

**BOOKING' on**

**Intranet**

**Note : Please do not try using the system available on intranet. This is just a case study.**

#### **Booking Instructions**

1. Invoke intranet by typing the URL <https://intranet.confBook.com>
2. Login using id and password
3. Intranet home page is displayed
4. Select Employee Corner and Click on Conference Room Booking option
5. 'My Bookings' option on the Conference Menu Page displays the View / Cancel Booking Page.
6. 'New Booking' option on the Conference Menu Page displays the Conference Booking Page.

#### **Making a new booking:**

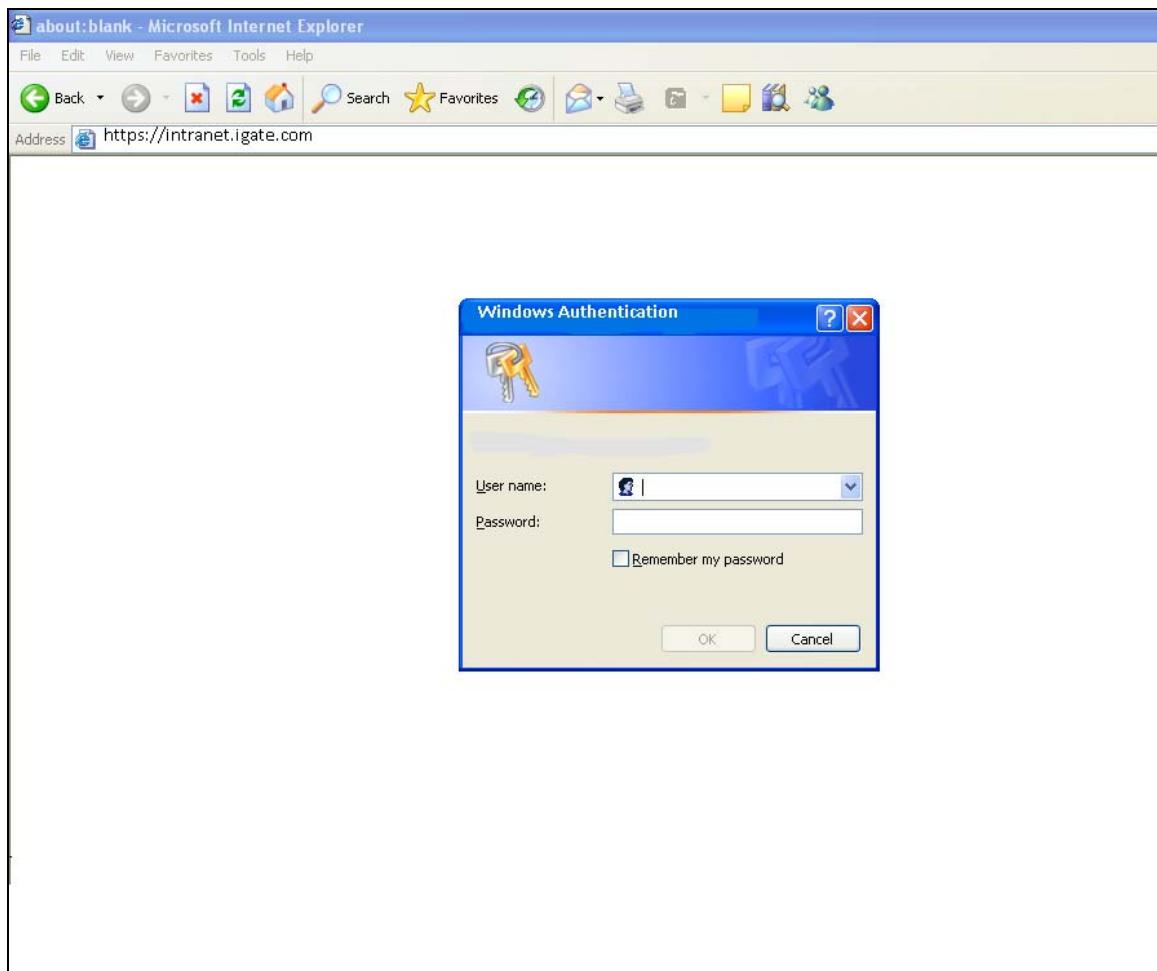
1. If you belong to a Non-GE Business Unit, you can book conference rooms only in Non-GE areas. However if you belong to a GE Business unit, you can book conference rooms at any location.
2. The employee Email id field is automatically filled with the email id of the person logged in.
3. Select the location from the location drop down box. This will automatically populate the sub-locations under the selected location in a new drop down box.
4. Select the sub-location from the sub-location drop down box. This will automatically populate the date drop down boxes.
5. Select the month from the month drop down box. Booking of conference rooms can be done only one month in advance. The date drop down boxes will be appropriately populated for this.
6. Select the date from the date drop down box.
7. Select the year from the year drop down box.
8. Upon selecting a valid location, sub location and date the complete Conference Room booking form is displayed
9. The rooms and devices available at the selected sub-location will be displayed in tabular format.

10. Enter your extension number and email id if it is not displayed correctly.
11. Select the room from the room drop down box.
12. Select the time span for which you want to book the conference room.
13. Select the communication and visual device if required.
14. Clicking 'Submit' button would validate whether the room and devices requested are available in the time span specified.
15. If either the room or devices requested are not available in the time span selected, an appropriate error message will be displayed else the booking will be registered. An Email will be sent to you as a confirmation for the same.

**Viewing / Cancellation of Bookings:**

1. All the booking made by you will be displayed. Bookings of current and future dates will only be displayed.
2. Select the bookings you wish to delete
3. Click the 'Delete' button to delete the selected bookings.

**Invoke <https://intranet.confBook.com>**



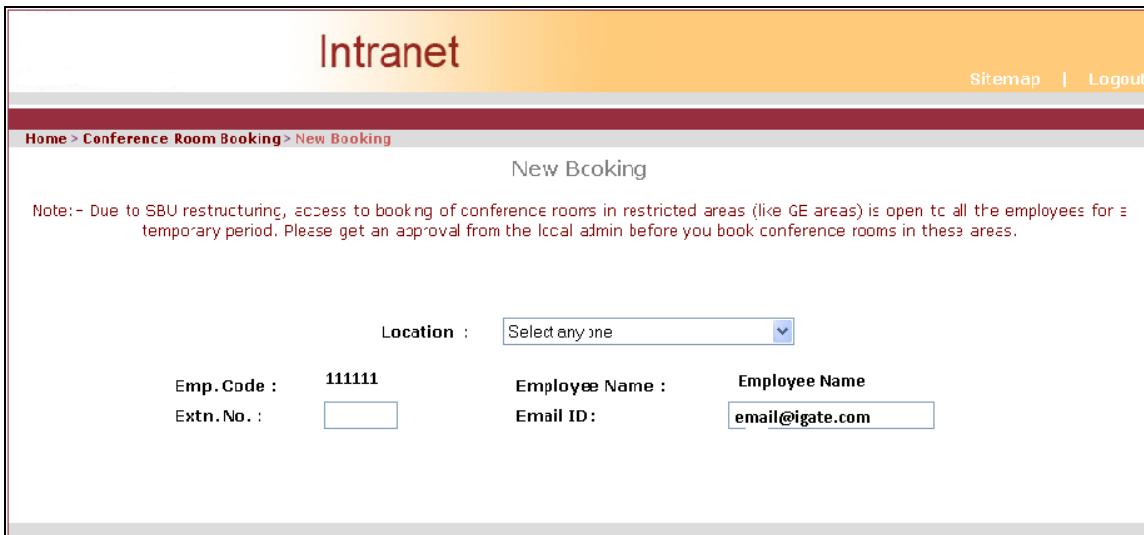
Intranet home → Employee Corner → Conference Room Booking



The screenshot shows the Capgemini Intranet homepage with a yellow header bar containing the word "Intranet". Below the header is a red navigation bar with "Home > Employee Self Service". A dark red sidebar on the left contains a "Menu" section with the following items:

- ◆ US Payroll - Electronic pay slip      Help
- ◆ Conference Room Booking
- ◆ Global HelpDesk
- ◆ EPFO-NSSA Allotment
- ◆ Library
- ◆ Manage your Account
- ◆ Parents Medical Insurance
- ◆ Compensation Letter
- ◆ List of Hospitals Across India
- ◆ Confidante- Register for Employee Counseling Services

Conference Room Booking → New Booking



The screenshot shows the "New Booking" page under "Conference Room Booking". The header includes "Intranet", "Sitemap", and "Logout". The breadcrumb navigation shows "Home > Conference Room Booking > New Booking". The main content area has a heading "New Booking" and a note: "Note:- Due to SBU restructuring, access to booking of conference rooms in restricted areas (like GE areas) is open to all the employees for a temporary period. Please get an approval from the local admin before you book conference rooms in these areas." Below the note are input fields for "Location" (dropdown menu), "Emp. Code" (111111), "Extn. No." (empty input box), "Employee Name" (empty input box), "Email ID" (email@igate.com), and "Employee Name" (empty input box).

**HOME→Conference Room Booking → New Booking**

**Intranet**

Sitemap | Logout

[Home](#) > [Conference Room Booking](#) > [New Booking](#)

### New Booking

Note:- Due to SBU restructuring, access to booking of conference rooms in restricted areas (like GE areas) is open to all the employees for a temporary period. Please get an approval from the local admin before you book conference rooms in these areas.

|                               |   |   |   |  |   |
|-------------------------------|---|---|---|--|---|
| Location :                    | <input type="text" value="SEEPZ"/>  | Sub Location :  | <input type="text" value="SDFI - First Floor"/> |  |   |
| Date (mm/dd/yyyy) :           | <input type="text" value="07"/> / <input type="text" value="07"/> / <input type="text" value="2005"/> |   |   |  |   |
| <b>AVAILABLE ROOMS</b>        |   |   |   |  |   |
| <b>Time From</b>              | <b>Time To</b>  | <b>Description</b>  |   |  |   |
| 00:00:01                      | 23:59:59  | Main conference room  |   |  |   |
| <b>AVAILABLE DEVICES</b>      |   |   |   |  |   |
| <b>Time From</b>              | <b>Time To</b>  | <b>Spider</b>   | <b>Speaker</b>                                  | <b>OHP</b>   | <b>LCD/Video Projector</b>  |
| 00:00:01                      | 23:59:59  | 1   | 1   | 1  | 1   |
| Emp. Code :                   |   | Employee Name :   |   | Jyoti Suresh                                       |   |
| Extn. No. :                   |   | Email ID :  |   | <input type="text" value="jyotisuresh@igate.com"/> |   |
| Select Room :                 |   | <input type="text" value="Main conference room"/>   |   |  |   |
| Description of the meeting :  |   | <input type="text"/>  |   |  |   |
| Time From (hh:mm):            |   | <input type="text" value="01"/> : <input type="text" value="00"/> <input type="text" value="AM"/> | Time To (hh:mm):                                |  | <input type="text" value="01"/> : <input type="text" value="00"/> <input type="text" value="AM"/> |
| <b>Communication Device :</b> |   |   | <b>Visual Device :</b>                          |  |   |
| Spider Phone                  | <input type="radio"/>   | OHP   | <input type="radio"/>                           |  |   |
| Speaker Phone                 | <input type="radio"/>   | LCD/Video Projector   | <input type="radio"/>                           |  |   |
| None                          | <input checked="" type="radio"/>  | None  | <input checked="" type="radio"/>                |  |   |

If booking is successful, the following screen is displayed

HOME → Conference Room Booking → New Booking

## Intranet

[Sitemap](#) | [Logout](#)

[Home](#) > [Conference Room Booking](#) > [New Booking](#) > [New Booking Successful](#)

Booking Success

The Conference Room has been booked for the time requested.

 Click Here for more Bookings.

The booking status is also shown under My Bookings as shown below:-

HOME → Conference Room Booking → Booking Status

## Intranet

[Sitemap](#) | [Logout](#)

[Home](#) > [Conference Room Booking](#) > [Booking status](#)

Conference Room Booking Status

BOOKING STATUS FOR EMPLOYEE 31608

To cancel any booking, put a check in the corresponding checkbox and click on delete.

| Del                      | Location | Sub-Location        | Date Of Booking<br>(mm/dd/yyyy) | Time From | Time To  | Room Description     | Name         | Extn. No. | Communication Device | Visual Device       |
|--------------------------|----------|---------------------|---------------------------------|-----------|----------|----------------------|--------------|-----------|----------------------|---------------------|
| <input type="checkbox"/> | SEEPZ    | SDFII - First Floor | 7/7/2005                        | 15:00:00  | 17:00:00 | Main conference room | Jyoti Suresh | 5204      | speaker              | LCD/Video Projector |

[Delete](#)

## Stretched Assignments

### 4.1 Customer Complaint Form

This is a customer complaint form for the bank customers, those who want to raise the bank related complaints.

### 4.2 Instructions

Write the test case for this Customer Complaint form, based on the requirements given below.

While the customer clicking on the submit button, if all are correct, the details should be registered in the database and generated complaint number should be displayed on the screen



The screenshot shows a web-based customer complaint form titled "CUSTOMER COMPLAINT FORM". At the top right, there is a small image of a smiling person wearing a headset. A status message at the top says: "You will receive a response shortly by SMS or E-mail To know the status of your request call on our Toll Free number 1800112211 or 18004". The form contains several input fields and dropdown menus. The fields include: Customer Type (Existing SBI customer), Account Number, Name Of Complainant, Branch Code (If other than Home Branch), Mobile number, Telephone No., E-mail, Category Of Complaints (dropdown menu showing "Data Received from Server!!!"), Products & Services (dropdown menu showing "Data Received from Server!!!"), Nature Of complaint (dropdown menu showing "Data Received from Server!!!"), and a text area for Please Give brief details of your complaint (Max 200 Chars). Below these fields is a CAPTCHA section with the text "MVBW" and a refresh icon. At the bottom of the form are "Submit" and "Reset" buttons.

### 4.3 Rules

- **Customer Type** : Existing SBI customer
- **Account Number** : Should be 11 digits
- **Branch Code** : should be 4 digits

- **Nature of Complaint** will be displayed based on the selection of Products & Services and Products & services will be displayed based on the selection of Category of Complaints. Refer the below table.

| Category of Complaints  | Products & Services  | Nature of Complaint   |
|-------------------------|----------------------|---|
| <b>General Banking</b>  | Branch Related       | <ul style="list-style-type: none"> <li>• No Response to queries</li> <li>• Single Windows not doing all transactions</li> </ul>                 |
|                         | Pass Book Related    | <ul style="list-style-type: none"> <li>• Error in passbook entries</li> <li>• Passbook not issued/Delayed</li> </ul>                            |
| <b>Deposits</b>         | Opening of Accounts  | <ul style="list-style-type: none"> <li>• Nominee Updation Not done</li> <li>• Delay in opening Accounts</li> </ul>                              |
|                         | Transfer of accounts | <ul style="list-style-type: none"> <li>• Delay in transfer of fixed Deposits</li> <li>• Others</li> </ul>                                       |
| <b>Internet Banking</b> | Pre Login Complaints | <ul style="list-style-type: none"> <li>• Username/Password provided by branch not functional</li> <li>• Transaction rights not given</li> </ul> |
|                         | Online Bill Payment  | <ul style="list-style-type: none"> <li>• Unable to view Bills</li> <li>• Unable to view payment History</li> </ul>                              |

- Details of complaint: Should be of 200 characters max.