

# Testing Concepts

## Lesson 00:

IGATE is now a part of Capgemini

People matter, results count.



©2016 Capgemini. All rights reserved.  
The information contained in this document is proprietary and confidential. For  
Capgemini only.

## Testing Concepts

### Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
	0.1D	NA		Content Creation
	0.1	NA		Review
May-2009		NA	Priya Rane	Material Revamp
June-2011	1.0	NA	Hema G.	Material Revamp
April-2014	1.1	NA	Dayanand Patil	Material Revamp
June-2016	2.0	NA	Neelima P.	Post-Integration Material Revamp



Copyright © Capgemini 2011. All Rights Reserved 3

## Course Goals and Non Goals

### ■ Course Goals

- At the end of this program, participants gain an understanding of Verification & Validation process in project
- Participants get an understanding of different testing approaches, techniques & types
- They also learn how to create effective test cases using the different testing techniques to get a good test coverage of a software application
- Participants get an understanding of Importance of monitoring progress in testing process & different project metrics

### ■ Course Non Goals

- This course does not cover automation process of testing.



## Pre-requisites

- None



Copyright © Capgemini 2011. All Rights Reserved 4

## Intended Audience

- Test Engineers, Software Engineers and Senior Software Engineers



## Day Wise Schedule

- Day 1
  - Lesson 1: Fundamentals of Testing
  - Lesson 2: Types of Testing Techniques & Test Case Design
- Day 2
  - Lesson 2: Types of Testing Techniques & Test Case Design (Cont.)
- Day 3
  - Lesson 2: Types of Testing Techniques & Test Case Design (Cont.)
- Day 4
  - Lesson 2: Types of Testing Techniques & Test Case Design (Cont.)
  - Lesson 3: Testing throughout the Software Life Cycle
- Day 5
  - Lesson 3: Testing throughout the Software Life Cycle (Cont.)
  - Lesson 4: Test Management & Test Case Execution
- Day 3
  - Lesson 5: Testing Metrics
  - Lesson 6: Tool Supporting for Testing



Copyright © Capgemini 2011. All Rights Reserved 8

## Table of Contents

- Lesson 1: Fundamentals of Testing
  - 1.1 Some Facts
  - 1.2 Introduction to Software Testing
  - 1.3 Software Testing - Definitions
  - 1.4 Need of Software Testing
  - 1.5 Error-Failure-Defect
  - 1.6 Causes of Software Defects
  - 1.7 Cost of Software Defects
  - 1.8 What does Software Testing reveal
  - 1.9 Importance of Software Testing
  - 1.10 Importance of Testing Early in SDLC Phases
  - 1.11 Testing and Quality
  - 1.12 Quality Perceptions



Copyright © Capgemini 2011. All Rights Reserved 1

## Table of Contents

- Lesson 1: Fundamentals of Testing
  - 1.13 Seven Testing Principles
  - 1.14 Economics of Testing
  - 1.15 How Testing is conducted?
  - 1.16 Software Testing – Then (Past)
  - 1.17 Software Testing – Now (Present)
  - 1.18 Scope of Software Testing
  - 1.19 Factors influencing the Scope of Testing
  - 1.20 Risk Based Testing
  - 1.21 Project Risks
  - 1.22 Product Risks
  - 1.23 Need of Independent Testing
  - 1.24 Activities in Fundamental Test Process



Copyright © Capgemini 2011. All Rights Reserved 8

## Table of Contents

- Lesson 1: Fundamentals of Testing

- 1.23 Attributes of a good Tester
- 1.24 Psychology of Testing
- 1.25 Code of Ethics for Tester
- 1.26 FS SBU: Focus on Testing
- 1.27 Testing Roles in iTEAMS
- 1.28 Limitations of Software Testing



Copyright © Capgemini 2011. All Rights Reserved 9

## Table of Contents

- Lesson 2: Types of Testing Techniques & Test Case Design
  - 2.1 Verification and Validation
  - 2.2 Types of Testing Techniques
  - 2.3 Static & Dynamic Testing Techniques
  - 2.4 Introduction to Static Testing Techniques
  - 2.5 Static Testing Techniques – Defects Detected & Benefits
  - 2.6 Review Process Success Criteria
  - 2.7 Introduction to Dynamic Testing
  - 2.8 Types of Dynamic Testing Techniques
  - 2.9 White Box Test Techniques
  - 2.10 Black Box Testing
  - 2.11 Static vs. Dynamic Testing
  - 2.12 A good Test Case



Copyright © Capgemini 2011. All Rights Reserved 11

## Table of Contents

- Lesson 2: Types of Testing Techniques & Test Case Design
  - 2.13 Test Case Lifecycle
  - 2.14 Test Case Design Techniques
  - 2.15 What is test data?
  - 2.16 Properties of Good Test Data
  - 2.17 Test Data team
  - 2.18 Test data lifecycle
  - 2.19 Requirement and Planning
  - 2.20 Request Process
  - 2.21 Test Data Creation Techniques
  - 2.22 Test Data From Production Data
  - 2.23 Test Data Life Cycle - Maintenance
  - 2.24 Test Data in STLC - Staggered with test case Design



Copyright © Capgemini 2011. All Rights Reserved 11

## Table of Contents

- Lesson 2: Types of Testing Techniques & Test Case Design
  - 2.25 Test data in STLC -Standalone phase between Test Case Design and Test Case Execution
  - 2.26 What is Positive Testing?
  - 2.27 Advantages/Limitations of positive testing
  - 2.28 What is negative testing?
  - 2.29 Advantages/Limitations of negative testing
  - 2.30 Positive & Negative test scenarios
  - 2.31 What is Basic test?
  - 2.32 Example on Basic test
  - 2.34 What is Alternate test?
  - 2.35 Example on Alternate test
  - 2.36 Importance of writing positive, negative, basic, alternate test while designing test cases
  - 2.37 Best practices for test case maintenance



Copyright © Capgemini 2011. All Rights Reserved 12

## Table of Contents

- Lesson 3: Testing throughout the Software Life Cycle
  - 3.1 Testing throughout the Software Life Cycle
  - 3.2 Introduction of SDLC and V-Model
  - 3.3 SDLC and V-Model
  - 3.4 Iterative Life Cycles
  - 3.5 Rapid Application Development
  - 3.6 Rational Unified Process (RUP) Phases
  - 3.7 RUP Phases and Disciplines
  - 3.8 Agile Development – Extreme Programming (XP)
  - 3.9 Testing Phases
  - 3.10 Introduction of Component Testing
  - 3.11 Component /Unit Testing
  - 3.12 Introduction of Integration testing



Copyright © Capgemini 2011. All Rights Reserved 13

## Table of Contents

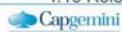
- Lesson 3: Testing throughout the Software Life Cycle
  - 3.13 Why Integration Testing is Required?
  - 3.14 Types of Integration testing
  - 3.15 Top Down Integration Testing
  - 3.16 Top Down Integration Testing
  - 3.17 Bottom Up Integration Testing
  - 3.18 Top Down vs. Bottom Up Testing
  - 3.19 Introduction to System Testing
  - 3.20 Types of System Testing



Copyright © Capgemini 2011. All Rights Reserved 14

## Table of Contents

- Lesson 4: Test Management & Test Case Execution
  - 4.1 Test Planning
  - 4.2 Test Plan Contents (IEEE 829)
  - 4.3 Test Planning Activities
  - 4.5 Entry Criteria for Functional Testing
  - 4.6 Test Case Execution - Pre-execution activities
  - 4.7 Types of Test Environment
  - 4.8 Before starting Execution
  - 4.9 Test Case Execution
  - 4.10 Exit Criteria for Functional Testing
  - 4.11 Test Estimation Techniques
  - 4.12 Factors affecting Test Effort
  - 4.13 Independent Testing
  - 4.14 Roles & Responsibilities - Working as Test Leader
  - 4.15 Roles & Responsibilities - Working as a Tester



Copyright © Capgemini 2011. All Rights Reserved 15

## Table of Contents

- Lesson 5: Testing Metrics
  - 5.1 Monitoring the Progress
  - 5.2 Metrics of Test Progress
  - 5.3 Reporting Test Status
  - 5.4 Test Control
  - 5.5 Configuration Management & Configuration Control
  - 5.6 Products for Configuration Management in Testing
  - 5.7 Definition of Metrics
  - 5.8 Need of Metrics
  - 5.9 Metrics for Testing
  - 5.10 Types of Metrics
  - 5.11 Types of Metrics – Project Metrics
  - 5.12 Types of Metrics – Process Metrics
  - 5.13 Types of Metrics – Productivity Metrics
  - 5.14 Types of Metrics – Closure Metrics



Copyright © Capgemini 2011. All Rights Reserved 16

## Table of Contents

- Lesson 6: Tool Supporting for Testing
  - 6.1 Tool support for Testing
  - 6.2 Test Tools Classification
  - 6.3 Tool Support for Management of Testing and Test
  - 6.4 Tool support for Static Testing
  - 6.5 Tool support for Test Specification
  - 6.6 Tool support for Test Execution & Logging
  - 6.7 Tool support for Performance & Monitoring
  - 6.8 Tool support for specific Testing Needs
  - 6.9 Need of Software Testing Tools
  - 6.10 Potential Benefits of using Tools
  - 6.11 Risks of using Tools
  - 6.12 Special Considerations for some Types of Tools
  - 6.13 Introducing a Tool into an Organization



Copyright © Capgemini 2011. All Rights Reserved 17

## 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/>



## Next Step Courses

- Automation testing



## **Testing Concepts**

Lesson 1: Fundamentals of  
Testing

## Lesson Objectives

- To understand the following topics:
  - Some Facts
  - Introduction to Software Testing
  - Software Testing - Definitions
  - Need of Software Testing
  - Error-Failure-Defect
  - Causes of Software Defects
  - Cost of Software Defects
  - What does Software Testing reveal
  - Importance of Software Testing
  - Importance of Testing Early in SDLC Phases
  - Testing and Quality
  - Quality Perceptions
  - Seven Testing Principles
  - Economics of Testing
  - How Testing is conducted?

Copyright © Capgemini 2012. All Rights Reserved.

## Lesson Objectives

- To understand the following topics:
  - Software Testing – Then (Past)
  - Software Testing – Now (Present)
  - Scope of Software Testing
  - Factors influencing the Scope of Testing
  - Risk Based Testing
  - Project Risks
  - Product Risks
  - Need of Independent Testing
  - Activities in Fundamental Test Process
  - Attributes of a good Tester
  - Psychology of Testing
  - Code of Ethics for Tester
  - FS SBU: Focus on Testing
  - Testing Roles in iTEAMS
  - Limitations of Software Testing

Copyright © Capgemini 2012. All Rights Reserved.

## Some Facts !!

Ariane 5 explosion - data conversion of a 64-bit no to 16-bit

Patriot missile - rounding error....Kills 28, injures 100

Mars Climate Orbiter lost - (Mixture of pounds and kilograms, 1999)

F16 autopilot - flipped plane upside down whenever it crossed the equator

Microsoft's anti-Unix site crashes  
News: [vnunet.com](http://vnunet.com)

Yahoo glitch strikes again  
03/20/2002



Copyright © Capgemini 2012. All Rights Reserved.

### Some Facts

Ariane 5 explosion - data conversion of a 64-bit no to 16-bit

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure.

Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

The internal SRI\* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer

Patriot missile - rounding error, kills 28, injures 100

A report of the General Accounting office, GAO/IMTEC-92-26, entitled Patriot Missile Defense: Software Problem Led to System Failure at Dhahran,

Saudi Arabia reported on the cause of the failure. It turns out that the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors.

Mars Climate Orbiter lost - (Mixture of pounds and kilograms, 1999)

The peer review preliminary findings indicate that one team used English units (e.g. inches, feet and pounds) while the other used metric units for a key spacecraft operation

F16 autopilot - flipped plane upside down whenever it crossed the equator

## Some Facts !!

Excel gives  $77.1 \times 850 = 100000$  instead of 65535

Y2K problem in Payroll systems designed in 1974

Disney's Lion King - 'Simba'



Copyright © Capgemini 2012. All Rights Reserved.

Microsoft's anti-Unix site crashes - News: vnu.com

The Web site launched by Microsoft and Unisys to lure customers away from

Unix has turned into a major embarrassment in more ways than one. First it was revealed that the site was powered by an open-source version of Unix and was running on the Apache Web server. So Microsoft switched to its Internet Information Server software and the site crashed.

Yahoo glitch strikes again - 03/20/2002

Users report irregular and missing content Parts of Yahoo were shut down on

Tuesday following software problems encountered in the merging of Yahoo Groups and Yahoo Clubs.

Excel gives  $77.1 \times 850 = 100000$  instead of 65535

Excel gives  $77.1 \times 850 = 100000$  instead of 65535 : While multiplying two numbers in MS-Excel and if the product equals 65535, it always gives the result 100000.

Y2K problem in Payroll systems designed in 1974

In 1974, when the first payroll was developed, to minimize the utilization of memory space, the year of the dates were stored in two digits instead of four digits i.e. 00 instead of 1900. But after 25 years in the yr. 2000, the question arose that how to store this. Will it be considered 1900 or 2000?



Copyright © Capgemini 2012. All Rights Reserved.

#### 9. Disney's Lion King – Simba

At the fall of 1994 Christmas, the Disney company came up with its first venture – a CDROM game for children with animated story of 'The Lion King Simba'. The sale was very huge. However it was a great loss to the Disney –

it

failed to work on most of the systems available in the market. The very next day on December 26th, Disney's customer care phones began to ring with calls from angry parents and crying children. Disney failed to properly test the game software on different PC models available in the market and as a result, the software worked only on few systems that were just like the one used by Disney programmers.

## Introduction to Software Testing

- Software Testing is the process of executing a program with the intent of finding errors as early as possible in SDLC
- It is a process used to help identify the correctness, completeness and quality of a developed computer software
- Software Testing helps in Verifying and Validating if the Software is working as it is intended to be working
- Testing is a process that helps in finding out how well the product works :
  - Aimed at finding defects
  - Aimed at demonstrating lack of quality
  - Aimed at demonstrating the gap between specifications and actual product
  - Aimed at building faith in the end product that gives advise on quality and risk



Copyright © Capgemini 2012. All Rights Reserved

### Introduction to 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.

Quality: The degree to which a component system or process meets specified requirements and/or user/customer needs and expectation.

## Software Testing - Definitions

- The process of executing a program (or part of a program) with the intention of finding errors - G. J. Myers
- Software testing is the process of testing the functionality and correctness of software by running it
- Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements - IEEE 83a
- The process of analyzing a system to detect the difference between existing and required conditions and to evaluate the feature of the system - IEEE/ANSI, 1983 [Std 829-1983]

Copyright © Capgemini 2012. All Rights Reserved.

## Need of Software 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



Time limited

- Because software is likely to have faults
- Because failures can be very expensive
- To contribute to the delivery of higher quality software product
- To detect the faults in the Software before User finds it
  - To know the reliability of the software
- Undetected errors are costly to detect at a later stage
- To satisfy users and to lower the maintenance cost



Copyright © Capgemini 2012. All Rights Reserved.

Why is testing becoming such a crucial activity?

Because applications are becoming very complex with n-tiers in an application. When one tests a program one adds value to it through improved quality and reliability.

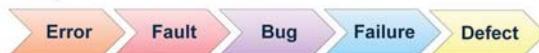
If not tested it can cause an unpleasant navigational error in case of a browsing applications or death or injury in case of safety critical applications. End customers are becoming more demanding & conscious about quality

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.

## Error-Failure-Defect

- Error(Mistake): A human action that produces an incorrect result
- Fault: A stage caused by an error which leads to unintended functionality of the program
- Bug: It is an evidence of the fault. It causes the program to perform in unintended manner. It is found before application goes into beta version
- Failure: Inability of the system to perform functionality according to its requirement
- Defect: It is a mismatch of the actual and expected result identified while testing the software in the beta version


Copyright © Capgemini 2014. All Rights Reserved.

### Error-Failure-Defect : Example

Consider the below program for addition of two integers;

```
#include<stdio.h>
1. int main ()
2. {
3.     int num1, num2, sum;
4.     num1 = 6;
5.     num2 = 4;
6.     sum = num1 - num2;
7.     printf("6 + 4 = %d", sum);
8. }
```

Output : 6 + 4 = 2

After compiling and running this program we see that the program has failed to do what it was supposed to do. The program was supposed to add two numbers but it did not. The result of  $4 + 6$  should be 10, but the result is 2. For now we have detected a **failure**. The **fault** in the program is the line 7 has '-' sign instead of '+' sign which led to a failure (deviation from the required functionality). In this case we can also say we have found the **bug**. **Error** is the mistake programmer made by typing '-' instead of '+' sign. The tester is testing the functionality of the program and realizes the output is faulty and will raise a **defect**.

**Software that does not work correctly can lead to many problems including loss of money, time, business reputation & could even cause injury or death.**

**Few examples :** Ariane 5 Space Program (\$7billion), Mariner space probe to Venus (\$250m), American Airlines (\$50m)

## Causes of Software Defects

- Software is written by human beings
  - Who know something, but not everything
  - Who have skills, but aren't perfect
  - Who do make mistakes (errors)
- Under increasing pressure to deliver to strict deadlines
  - No time to check but assumptions may be wrong
  - Systems may be incomplete
- Environmental conditions
  - Radiations, Magnetism, Electronic fields and pollution can cause faults in firmware or influence execution of software by changing hardware conditions
- Minimal or no proper documentation of Business Requirements
- Insufficient time window for development
- Lack of domain knowledge
- Programming Language constraints

Copyright © Capgemini 2012. All Rights Reserved. 11

## Cost of Software Defects

- It is Easy to find and fix defect in early stages rather than in the later phases of software.

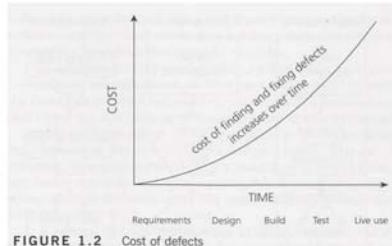


FIGURE 1.2 Cost of defects



Copyright © Capgemini 2012. All Rights Reserved. 12

### Cost of Software Defects

The cost of fixing a bug (defect) and making the required changes in early phases of software development is less as compared to the same detected in later phases since cost is spent at four different times in a SDLC;

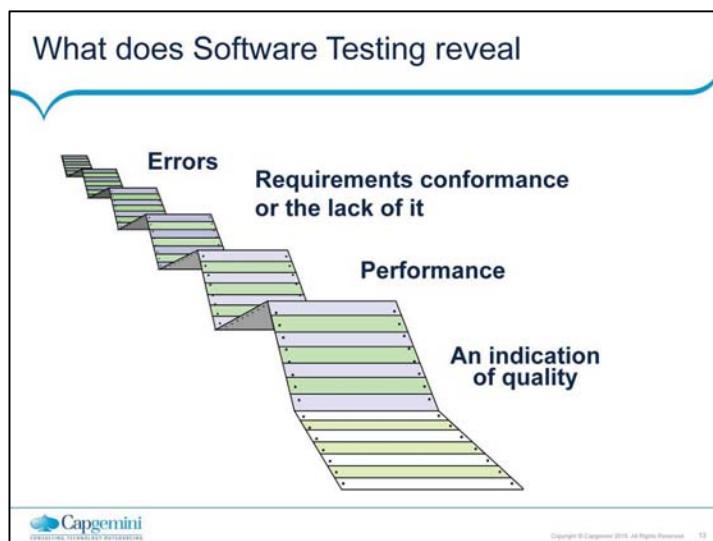
First, cost is spent in writing wrong specifications, erroneous coding and incorrectly documenting the system

Second, cost is spent on detecting the errors rather on preventing errors

Third, cost is spent on removing discovered errors in specifications, coding, and documentation

Fourth, cost is spent on re-testing the system to determine whether the errors have been fixed.

Therefore, to achieve lower cost and high quality systems, testing must not be considered as single phase activity; it must be incorporated in all SDLC phases.



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.

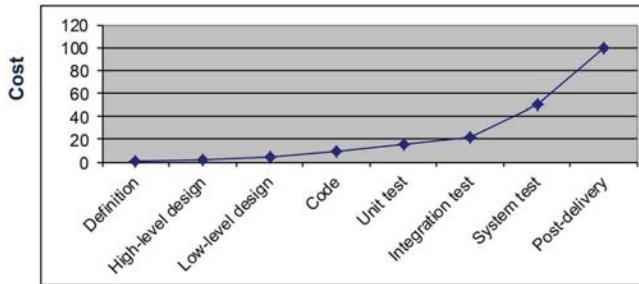
## Importance of Software Testing

- Ensures that the product is usable
- Ensures that Customer's Objectives are met
- Early detection of errors to prevents breakdown at a later stage
- Ensures that the software is reliable
- Builds Confidence in software
- Increases Customer Satisfaction
- Ensures effective execution in the given environment
- Reduces overall cost of software
- Reduces time for going live (production)

Copyright © Capgemini 2012. All Rights Reserved. 14

## Importance of Testing Early in SDLC Phases

- Error removal cost over SDLC



## Importance of Testing Early in SDLC Phases

- Prevents future Problems thus lowering the cost
- Testing will not be a bottleneck anymore
- Testers become more familiar with the software, as they are more involved with the evolution of the product.
- Reduces the chances of failure
- The test environment can be prepared in advance
- The risk of having a short time for testing is greatly reduced
- Maintains "quality culture" in the organization



Copyright © Capgemini 2012. All Rights Reserved. 10

### Importance of Testing Early in SDLC Phases

Many problems raise during planning or design. Requirements testing can prevent future problems thus lowering the cost. Since the testing process is involved with all phases of the SDLC, Management will not feel as if testing is a bottleneck to release the product.

Test cases written during requirements and shared with the Dev. team before the construction phase can help developers to reduce the chances of failure

The test environment can be prepared in advance

The risk of having a short time for testing is greatly reduced Involving quality assurance in all phases of the SDLC helps creating a 'quality culture' inside the organization.

## Testing and Quality

- Testing helps to measure the Quality of software in terms of
  - Number of defects found
  - important information regarding the Non functional attributes like Reliability, Security, Performance, etc.
- Quality Definition
  - Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected from all professionally developed software.



Copyright © Capgemini 2012. All Rights Reserved. 17

### Importance of Software Quality

Aim : Customer & User Satisfaction

Quality is a competitive issue

Quality is must for survival

Quality gives entry into International market.

Quality is cost effective

Quality helps retain customers

## Quality Perceptions

- Engineer may judge :
  - User satisfaction
  - Portability
  - Maintainability
  - Robustness & Efficiency
- Customer may judge :
  - Cost
- User may judge :
  - Reliability
  - usability



Copyright © Capgemini 2012. All Rights Reserved. 18

## Seven Testing Principles

- Principle 1 - Testing shows presence of defects but cannot prove that there are no defects
- Principle 2 - Exhaustive testing is impossible
- Principle 3 - Early testing
- Principle 4 - Defect Clustering
- Principle 5 - Pesticide Paradox
- Principle 6 - Testing is context dependent
- Principle 7 - Absence of Errors fallacy

Copyright © Capgemini 2012. All Rights Reserved. 19

### Testing Principles

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. Testing shows presence of defects but cannot prove that there are no defects. Exhaustive testing is impossible. Risk analysis & prioritisation are used to focus testing effort.

**Early testing :** Testing activity is started as early as possible in the SDLC to find defects early

**Defect Clustering :** Testing effort should be focused proportionally to the expected or later observed defect density of modules. A small number of modules usually contains most of the defects.

**Pesticide Paradox :** If same set of tests are repeated, then NO defects found. To overcome this problem, test cases need to be reviewed and revised regularly to assess different parts of the software or systems.

**Testing is context dependent :** Safety Critical Software is tested differently compared to an e-commerce application

**Absence of Errors fallacy:** Finding & fixing of errors does not help if the system built is unusable and does not fulfil the users' needs and expectations.

## Economics of Testing

- Economics of Testing
  - It is both the driving force and the limiting factor
- Driving - Earlier the errors are discovered and removed in the lifecycle, lowers 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 2012. All Rights Reserved. 

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.

## 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 2012. All Rights Reserved. 21

### How testing is conducted?

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

## Software Testing – Then (Past)

- Testing led to blame each other and giving faulty justifications and excuses which would arise quarrels among the team members
- Not much re-work was welcomed
- Developers misunderstood that increased testing would increase the project cost unnecessarily
- It was understood that software has quality if it is just easy to maintain, reusable and flexible.
- Testing was conducted only at project execution phase
- Ad-hoc & need driven
- Totally manual
- Testing jobs perceived as low scale as compared to other discipline

Copyright © Capgemini 2014. All Rights Reserved. 12

## Software Testing – Now (Present)

- Organized body of knowledge
  - Independent Testing teams
  - Testing Models
  - Testing Knowledge Groups
- A specialized engineering discipline in great demand
- Defects are highlighted and brought to surface so as to conduct corrective measures and achieve user satisfaction
- Testing lead to cooperative solutions so as to prevent the failures
- Documentation is considered as essential to note down the lessons learnt so that mistakes are not repeated
- Developers know that testing increases the business profit
- Describes software quality in terms of integrity, reliability, usability and accuracy
- Testing is conducted as the project initiates

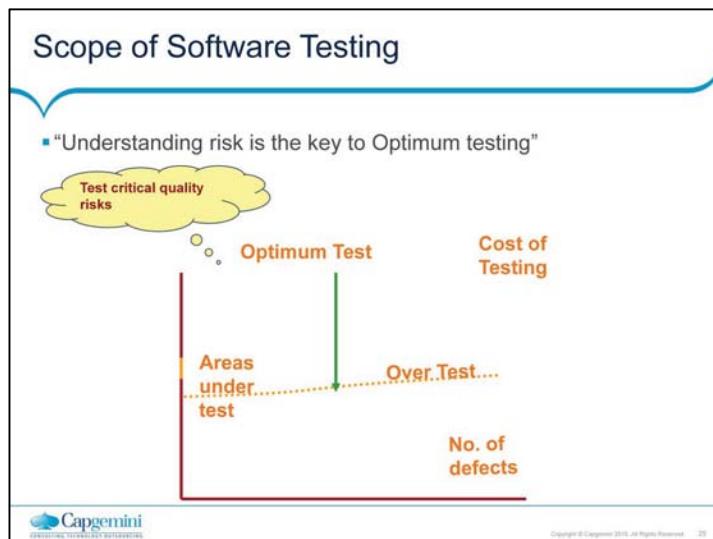
Copyright © Capgemini 2012. All Rights Reserved. 27

## Scope of Software Testing

- Bad news : You can't test everything
- Good news : There is such a thing as "good enough"
- Bad news : Good enough may cost too much
  
- What do we do : Increase focus via systematic process of elimination
  
- What you might test?
  - Those areas which are within the scope of your project
- What you should test?
  - The critical system functionality which effects the customers & users experience of quality
- What you can test?
  - Estimate time & resource for risk driven effort



Copyright © Capgemini 2012. All Rights Reserved. SP



## Factors influencing the Scope of Testing

- Contractual requirements
- Legal requirements
  - Privacy related laws
  - Non-disclosure of identity
- Industry-specific requirements
  - Aircraft safety equipment
- Scope of Testing is about identifying the correct test cases for automation
- The steps involved are:
  - Identify various factors that form the basis of identifying the candidate test cases
  - Apply 'Divide & rule' strategy : Break the application into smaller modules
  - Analyze each module to identify the candidate test cases
  - Calculate ROI
- Factors influencing the scope of testing :
  - In small projects
    - Test case writing
    - Test case execution
    - Regression testing
    - In Large projects
      - Setting up test bed
      - Generating test data, test scripts, etc.



Copyright © Capgemini 2012. All Rights Reserved. 10

## Risk Based Testing

- Risk:

- A factor that could result in negative consequences; usually expressed as impact and likelihood
- Risks are used to decide where to start testing and where to test more.
- Risk Based testing:
- Testing oriented towards exploring and providing information about product risks
- Risk based Testing is used to reduce risk of adverse effect occurring or to reduce the impact of adverse effect
- It draws on the collective knowledge and insight of the project stakeholders to determine the risk and the level of testing required to address those risks.



Copyright © Capgemini 2012. All Rights Reserved. 17

Risk identified in the Risk Based testing is used to:

Determine the test techniques to be employed

Determine the extent of testing to be carried out

Prioritize testing in an attempt to find the critical defect as early as possible

Determine whether any non testing activities could be employed to reduce risk

Risk Management activities provide a discipline approach to minimize the product failure:

Assess and reassess what can go wrong (risks)

Determine what risks are important to deal with

Implement action to deal with those risks

## Project Risks

- Project Risk

- A risk related to management and control of the (test) project is called as Project Risk

- Project risk are:

- Organizational factor

- Skill, training and staff shortage
    - Personnel issues
    - Political issues

- Technical issues

- Problem in defining right requirements and quality of the design code, test data and test
    - The extent to which requirement cannot be met given existing constraints
    - Test environment not ready on time or late data conversion, migration planning and development and testing data conversion/migration tools

- Supplier issues

- Failure of a third party
    - Contractual issues



Copyright © Capgemini 2012. All Rights Reserved. 19

## Product Risks

- Product Risk: it is directly related to the test object
- Risks related to quality of a product:
  - Failure-prone software delivered
  - The Potential that the software/hardware could cause harm to an individual or company
  - Poor software characteristics
  - Poor data integrity and quality
  - Software that does not perform its intended functions



Copyright © Capgemini 2014. All Rights Reserved. 29

## Need of Independent Testing

- Unbiased testing is necessary to objectively evaluate quality of a software
- Developer carrying out testing would not like to expose defects
- Assumptions made are carried into testing
- People see what they want to see.
- More effective in terms of Quality & Cost
- It is conducted by an independent test team other than developer to avoid author bias and is more effective in finding defects and failures
- The tester sees each defect in a neutral perspective
- The tester is totally unbiased
- The tester sees what has been built rather than what the developer thought
- The tester makes no assumptions regarding quality

Copyright © Capgemini 2014. All Rights Reserved. 10

## Activities in Fundamental Test Process

- Test planning and control
  - It defines the objectives and specification of test activities
  - Test control is the on going activity of comparing actual progress against the plan
- Test analysis and design
  - Testing objectives are transformed into tangible test conditions and test cases
  - Reviewing the test basis
  - Evaluating testability of the test basis and test objects
  - Identifying, Designing and Prioritizing test conditions based on analysis of test item, the specification, behaviour and structure of the software
  - Designing and Prioritizing high level test cases
  - Identifying necessary test data
  - Designing test environment setup and identifying required infrastructure and tools
  - Creating bi-directional traceability between test basis and test cases

Copyright © Capgemini 2012. All Rights Reserved. 17

## Activities in Fundamental Test Process

### ▪ Test implementation and execution

- Test procedures (scripts) are specified by combining test cases in a particular order and the environment is set up and tests are run
- Finalizing, implementing and prioritizing test cases
- Developing and prioritizing test procedures, creating test data and writing automated test scripts
- Creating test suites from test procedures for efficient test execution
- Verification if Test environment is setup correctly
- Verifying and updating bi-directional traceability between test basis and test cases
- Executing test procedure using tool or manually according to the planned sequence
- Logging the outcome of the test execution and recording the identities and version of the software under test tools and test ware
- Comparing actual result with expected result
- Reporting discrepancies and analysing their root cause
- Repeating test activities as a result of action taken for each discrepancy

Copyright © Capgemini 2012. All Rights Reserved. 10

## Activities in Fundamental Test Process

- Evaluating exit criteria and reporting
  - Test execution is assessed against the defined objectives
  - Test logs are checked against the exit criteria specified in test planning
  - Assessment is done if more tests are needed
  - Test summary report is written
- Test closure activities
  - Data from completed test activity is collected to consolidate experience, facts and numbers
  - This is done at milestones



Copyright © Capgemini 2012. All Rights Reserved. 10

Testware: Artifacts produced during the test process required to plan, design and execute test and any additional software or utilities used in testing.

## Attributes of a good Tester

- A good test engineer has a 'test to break' attitude
- Need to take a different view, a different mindset ("What if it isn't?", "What could go wrong?")
- An ability to understand the point of view of the customer
- A passion for quality and attention to detail
- Notice little things that others miss/ignore (See symptom not bug)
- Ability to communicate fault information to both technical (developers) and non-technical (managers and customers)
- Tact and diplomacy for maintaining a cooperative relationship with developers
- Work under worst time pressure (at the end)
- "Patience"

Copyright © Capgemini 2012. All Rights Reserved. IS

## 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 2012. All Rights Reserved. 10

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.

The mindset to be used while testing and reviewing is different from that used while developing software.

Separation of responsibilities of development and testing are done to help focus efforts and provide an independent / unbiased view.

While a certain level of independence often makes the tester more effective at finding defects and failures, this independence can however not replace familiarity which developers possess.

People and projects are driven by objectives. For example, to find defects and confirm that the software meets its objectives. It is therefore important to clearly state the objectives of testing.

Identifying failures during testing may be perceived as criticism against the product and its author. Testing is therefore often viewed as a destructive activity.

Communication of errors in the product in a constructive way therefore assumes particular importance in order to make testing appear constructive and supportive.

The Test Leaders and Testers therefore need to have good interpersonal and communication skills to overcome this difference of perception.

Some ways of improving communication and relationships between testers and others:

- Start with collaboration rather than battles – Common goal of better quality system
- Communicate findings in a neutral, fact-focussed way, without criticising individuals who created the tested products
- Try to understand what the other person feels and why they react in the way they do
- Confirm that the other person has understood what you conveyed.

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 not scalable, 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.

## Code of Ethics for Tester

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is therefore necessary, among other reasons to ensure that the information is not put to inappropriate use.

- PUBLIC - Tester shall act consistently with public interest
- CLIENT AND EMPLOYER - Tester shall act in best interests of their client and employer
- PRODUCT - Tester shall ensure that the deliverables they provide meet highest professional standards possible
- JUDGMENT - Tester shall maintain integrity and independence in their professional judgment
- MANAGEMENT - Tester managers and leaders shall subscribe to and promote an ethical approach to manage software testing
- PROFESSION - Tester shall advance the integrity and reputation of the profession consistent with the public interest
- COLLEAGUES - Tester shall be fair to and supportive of their colleagues, and promote cooperation with software developers
- SELF - Tester shall participate in lifelong learning and shall promote an ethical approach to the practice of the profession

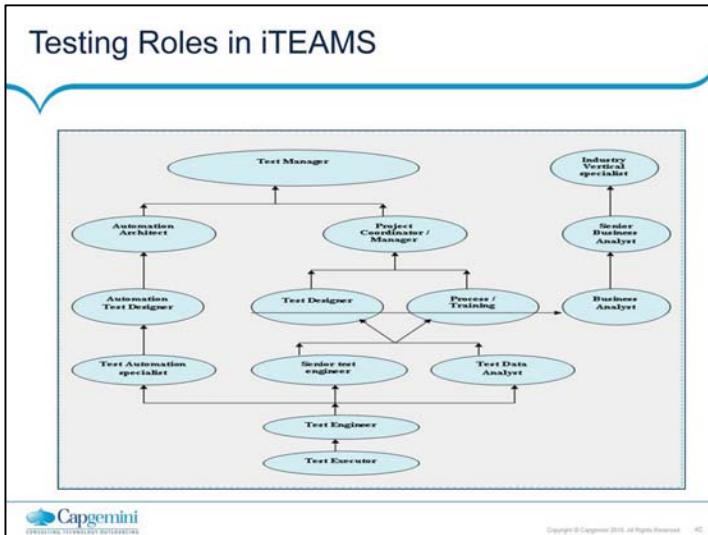


Copyright © Capgemini 2014. All Rights Reserved. 10

## FS SBU: Focus on Testing

- Fastest growing service line is FS SBU (Financial Services Strategic Business Unit)
- Team of 1600+ test engineers
- Was CMM level 5 assessed within a year of being established
- Presence in most of the existing FS SBU accounts
- High Focus on competency building
- Do almost all types of testing including Performance monitoring and Public Website testing
- Offering Security Testing

Copyright © Capgemini 2014. All Rights Reserved. 10



Copyright © Capgemini 2012. All Rights Reserved. 40

## 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 2012. All Rights Reserved. #1

## 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
  - Scope of testing
  - 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
  - Various principles of testing



Copyright © Capgemini 2014. All Rights Reserved. 42

## Review Question

- Question 1: What is visible to end-users is a deviation from the specific or expected behavior is called as
  - Defect
  - Bug
  - Failure
  - fault
- Question 2: \_\_\_\_\_ is a planned sequence of actions
- Question 3: Pick the best definition of Quality :
  - Quality is job done
  - Zero defects
  - Conformance to requirements
  - Work as designed
- Question 4: One cannot test a program completely to guarantee that it is error free
  - Option: True / False
- Question 5: One can find missing paths due to coding errors
  - Option: True / False

Copyright © Capgemini 2012. All Rights Reserved. 43

## Review Question: Match the Following

1. 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                           |

Copyright © Capgemini 2012. All Rights Reserved.

## Testing Concepts

Lesson 2: Types of Testing  
Techniques & Test Case  
Design

## Lesson Objectives

- To understand the following topics:
  - Verification and Validation
  - Types of Testing Techniques
  - Static & Dynamic Testing Techniques
  - Introduction to Static Testing Techniques
  - Static Testing Techniques – Defects Detected & Benefits
  - Review Process Success Criteria
  - Introduction to Dynamic Testing
  - Types of Dynamic Testing Techniques
  - White Box Test Techniques
  - Black Box Testing
  - Static vs. Dynamic Testing
  - A good Test Case
  - Test Case Lifecycle
  - Test Case Design Techniques



## Lesson Objectives

- To understand the following topics:
  - What is test data?
  - Properties of Good Test Data
  - Test Data team
  - Test data lifecycle
  - Requirement and Planning
  - Request Process
  - Test Data Creation Techniques
  - Test Data From Production Data
  - Test Data Life Cycle - Maintenance
  - Test Data in STLC - Staggered with test case Design
  - Test data in STLC -Standalone phase between Test Case Design and Test Case Execution



## Lesson Objectives

- To understand the following topics:
  - What is Positive Testing?
  - Advantages/Limitations of positive testing
  - What is negative testing?
  - Advantages/Limitations of negative testing
  - Positive & Negative test scenarios
  - What is Basic test?
  - Example on Basic test
  - What is Alternate test?
  - Example on Alternate test
  - Importance of writing positive, negative, basic, alternate test while designing test cases
  - Best practices for test case maintenance



## 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.



Copyright © Capgemini 2010. All Rights Reserved.

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.

## Verification and Validation (cont.)

### ▪ 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 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.



Copyright © Capgemini 2010. All Rights Reserved.

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

## Types of Testing Techniques

- Static Testing

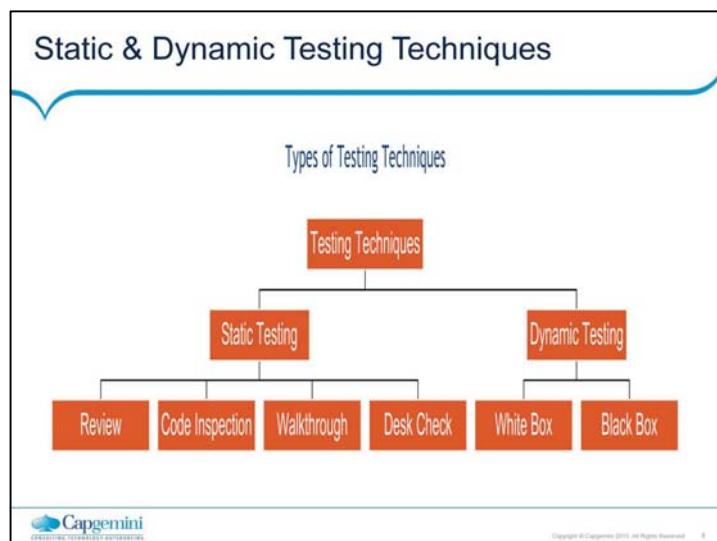
- It is a **verification** process
- Testing a software without execution on a computer. Involves just examination/review and evaluation
- It is done to test that software confirms to its SRS i.e. user specified requirements
- It is done for **preventing** the defects

- Dynamic Testing

- It is a **validation** process
- Testing software through executing it
- It is done to test that software does what the user really requires
- It is done for **detecting** the defects



Copyright © Capgemini 2010. All Rights Reserved.



## Introduction to Static Testing Techniques

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

Copyright © Capgemini 2010. All Rights Reserved.

### Introduction to Static Testing Techniques

How can we evaluate or analyze a requirements document, a design document, a test plan, or a user manual or examine a source code?

By reviewing those.

These static techniques rely on manual examinations (Reviews) and automated Analysis (Static Analysis) without execution

Any software product can be reviewed

Purpose of reviews

Finding defects

informational, communicational and educational

## 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 2010. All Rights Reserved.

### 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?

## 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?
- 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?


Copyright © Capgemini 2010. All Rights Reserved.

### 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?

## 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 2010. All Rights Reserved.

Moderator duties includes:

Distribution materials,  
Scheduling the Inspection Session  
Leading the session  
Recording all errors found  
Ensures that the errors found are subsequently corrected.

## Code Inspection

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



Copyright © Capgemini 2010. All Rights Reserved.

13

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.

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

Copyright © Capgemini 2010. All Rights Reserved.

Walkthrough meeting.

This meeting will include following members:

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

Walk through procedure

The designer simulates the program.

She/he shows, step by step what the program will do with the test data supplied by the reviewers.

The simulation shows how different pieces of the system interact and can expose awkwardness, redundancy and many missed details.

Different from inspection that it needs the participants to be ready with test cases.

## Desk Checking (Peer Review)

- 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 2010. All Rights Reserved. 10

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.

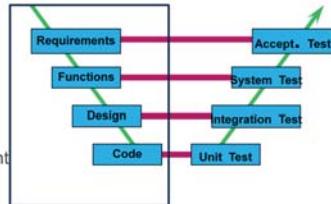
E.g. Two programmers might swap the program rather than desk checking their own programs.

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

## Static Testing Techniques – Defects Detected & Benefits

- Types of defects found during static reviews:

- Deviations from standard
  - Requirement defects
  - Design defects
  - Insufficient maintainability
  - Incorrect interface specification
- Benefits of Reviews:
- Early feedback on quality
  - Development productivity improvement
  - Reduced development timescales
  - Reduced testing time and cost
  - Lifetime cost reductions
  - Reduced fault levels
  - Increased awareness of quality issues

Copyright © Capgemini 2010. All Rights Reserved.

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.

## Review Process Success Criteria

Success factors for reviews:

- Review should be with clear objective
- Defects found are welcomed and expressed objectively
- Management supports a good review process
- The emphasize is on learning and process improvement
- The right people for review are involved
- Appropriate use of review techniques
- Reviews are conducted in a fair & trustworthy atmosphere
- Explicitly planning and tracking review activities
- Train the participants in the review techniques – particularly the formal ones such as inspection



Copyright © Capgemini 2010. All Rights Reserved. 43

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.

## Introduction to Dynamic Testing

- Dynamic Testing involves working with the software, giving input values and validating the output with the expected outcome
- Dynamic Testing is performed by executing the code
- It checks for functional behavior of software system , memory/CPU usage and overall performance of the system
- Dynamic Testing focuses on whether the software product works in conformance with the business requirements
- Dynamic testing is performed at all levels of testing and it can be either black or white box testing



Copyright © Capgemini 2010. All Rights Reserved.

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.

## Types of Dynamic Testing Techniques

- 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 2010. All Rights Reserved. 19

## 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

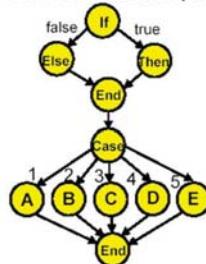


Copyright © Capgemini 2010. All Rights Reserved.

## 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



Copyright © Capgemini 2010. All Rights Reserved.

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

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

## 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 2020. All Rights Reserved.

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

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

## 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; }
}
```

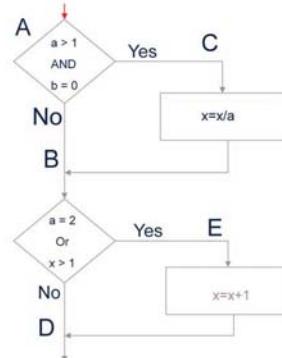


Copyright © Capgemini 2010. All Rights Reserved.

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

## 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.



Copyright © Capgemini 2010. All Rights Reserved.

24

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.

## 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 2010. All Rights Reserved.

## Decision Coverage

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

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

The flowchart illustrates a decision coverage scenario. It starts at decision point A, which checks if  $a > 1$  AND  $b = 0$ . If the answer is No, it proceeds to decision point B, which checks if  $a = 2$  OR  $x > 1$ . If the answer is No, it leads to terminal point D. If the answer is Yes, it leads to action block E, where  $x = x + 1$ , and then to decision point B. From decision point B, if the answer is Yes, it leads to action block C, where  $x = x/a$ , and then to terminal point C.

Capgemini  
Engineering Services India Pvt. Ltd.

Copyright © Capgemini 2012. All Rights Reserved. 30

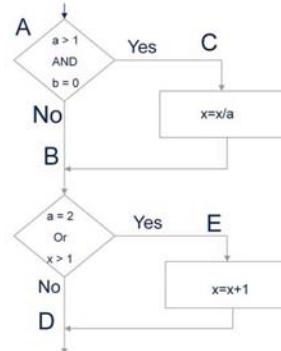
Decision coverage can cover two test cases covering paths ACE and ABD. Even if the above test cases satisfy decision coverage it still does not cover the path ACD and path ABE. Hence decision coverage though stronger criteria than statement it is still weak. There is only 50 percent chance that we would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

## 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)


Copyright © Capgemini 2010. All Rights Reserved.

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  $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ .

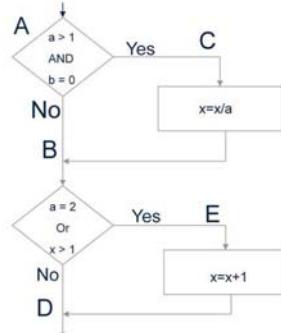
A compound condition is composed of two or more simple conditions, Boolean operators, and parentheses.

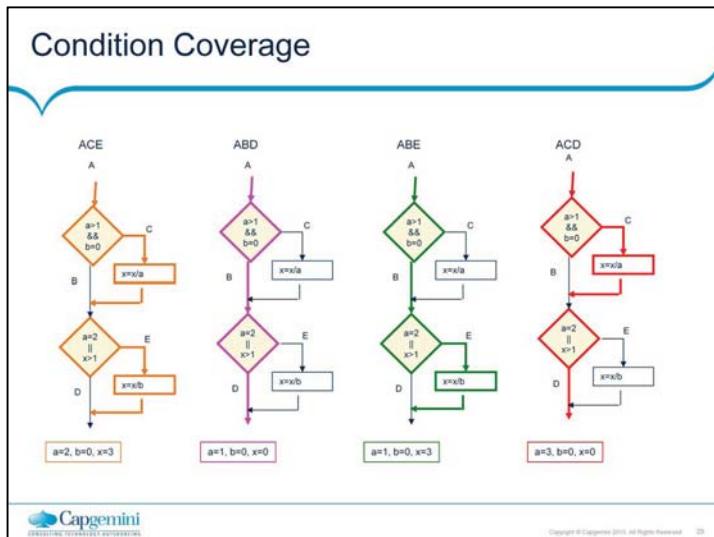
Condition coverage focuses on testing each condition in a program. The purpose of the condition testing is to detect not only errors in the conditions of a program but also other errors in the program.

## 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 2010. All Rights Reserved.



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.

## 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

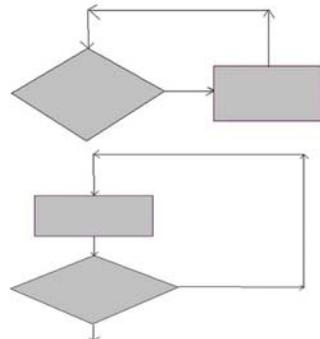


Copyright © Capgemini 2021. All Rights Reserved.

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

## Loop Testing

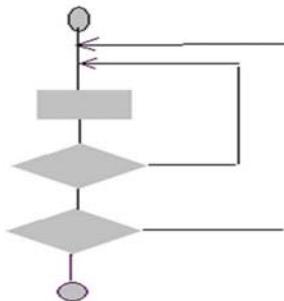
- Simple Loop Testing Procedure:
  - skip the entire loop
  - only one pass through the loop
  - make 2 passes through loop
  - m passes through loop where m<n
  - n-1, n, n+1 passes through the loop
- Where n is the maximum number of allowable passes through the loop



Copyright © Capgemini 2010. All Rights Reserved. 33

## 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

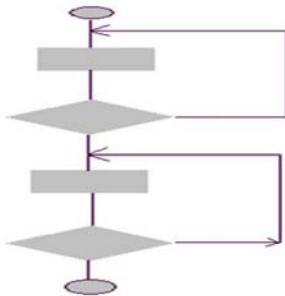


Copyright © Cengage 2009. All Rights Reserved.

Number of test cases increases as the loop gets extended.

## 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

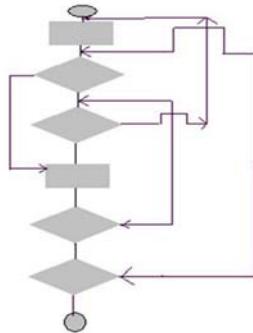


Copyright © Capgemini 2012. All Rights Reserved.

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.

## Loop Testing

- Spaghetti loops Testing Procedure:
  - Redesign using structured constructs



Copyright © Capgemini 2010. All Rights Reserved. 59

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

## 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



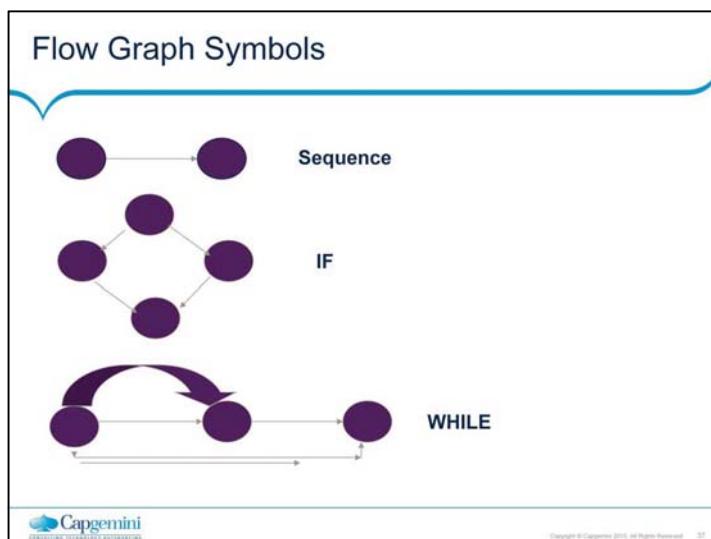
Copyright © Capgemini 2010. All Rights Reserved.

## 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 is called a predicate node



Copyright © Capgemini 2010. All Rights Reserved.



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.

## Cyclomatic Complexity

- Cyclomatic Complexity (Code 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 2010. All Rights Reserved.

Introduced by Thomas McCabe in 1976:

Count the regions of the flow graph (including the exterior)  
Or compute by  $e-n+2$  This is called Cyclomatic Complexity  
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.

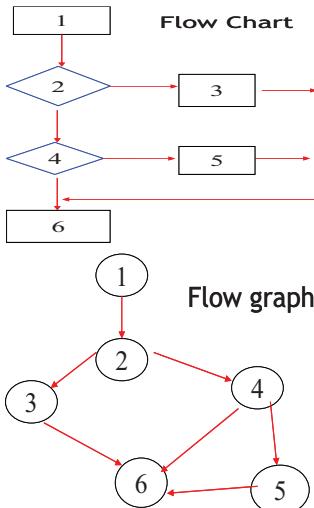
When used in context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us the upper bound of the number of tests that must be conducted to ensure that all statements have been executed at least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

## 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


Copyright © Capgemini 2010. All Rights Reserved.
09

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.

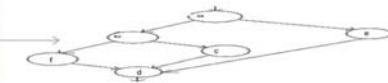


A flow chart depicts program control structure.  
Flow chart is mapped into flow graph.  
A sequence of process boxes and a decision diamond are map into a single node.

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.

## Calculating Cyclomatic Complexity : Example

In the given figure a and b are predicate nodes



1. Cyclomatic Complexity,  $v(G)$  for a flow Graph G is  $v(G) = E - N + L$

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  $V(G) = 7-6+2 = 3$

2.  $V(G)$  can also be calculated as  $V(G) = P+1$ , where P is the number of predicate nodes. Here  $V(G) = 2+1 = 3$

3. Also  $V(G)$  can be calculated as  $V(G) = R$  hence  $V(G) = 3$

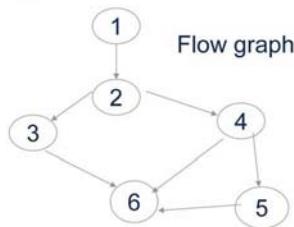


Copyright © Capgemini 2010. All Rights Reserved.

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.

## Calculating Cyclomatic Complexity : Example



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.



Copyright © Capgemini 2010. All Rights Reserved. 63

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 are required which traversed thorough the above paths.

## 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 2010. All Rights Reserved. 43

## 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 2010. All Rights Reserved. 43

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!  
}
```

## 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 2010. All Rights Reserved. 69

### 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.

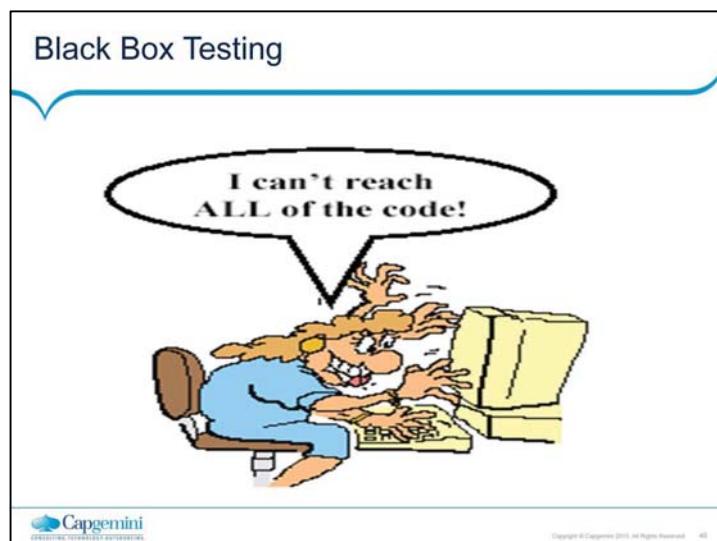
## Black Box Testing

- 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 2010. All Rights Reserved.

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

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



Trainer Notes

## Black Box Testing

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 2010. All Rights Reserved. 47

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  
incorrect or missing functions  
interface errors  
errors in data structures or external database access  
behavior or performance errors  
initialization errors.

## Black Box Test Techniques

There are various techniques to perform Black box testing ;

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



Copyright © Capgemini 2012. All Rights Reserved. 69

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

## 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

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 2012. All Rights Reserved. 40

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.

## 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 2010. All Rights Reserved.

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.

## Equivalence Partitioning

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



Identify - one valid equivalence class ( $1 < \text{count} < 999$ )

- two invalid equivalence classes ( $\text{count} < 1$ ) & ( $\text{count} > 999$ )



Copyright © Capgemini 2010. All Rights Reserved.

Equivalence classes may be defined according to the following guidelines.

If an input condition specifies a range, one valid and two invalid equivalence classes are defined.

If an input condition requires a specific value, one valid and two invalid EC are defined.

If an input condition specifies a member of a set, one valid and one invalid EC are defined.

In an input condition is Boolean, one valid and one invalid class are defined.

## 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 2010. All Rights Reserved.

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

## 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 2010. All Rights Reserved.

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.

## 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



Copyright © Capgemini 2010. All Rights Reserved.

### General guidelines:

- If an input or output condition specifies a range of values, write test cases for the ends of the range.
- If an input or output condition specifies a number of values, write test cases for the minimum and maximum number of values.
- If the input or output of a procedure is an ordered set, focus attention on the first and last elements of the set.

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

- 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.
- 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.
- 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.
- 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.

## 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



Copyright © Capgemini 2010. All Rights Reserved.

10

## 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, 32768	-32769,32768
B	-32768, -32767, 32767, 32766	-32769,32768



Copyright © Capgemini 2010. All Rights Reserved.

## 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 success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to lurk
- This is why error guessing testing technique should be used along with other formal testing techniques
- The basis behind this approach is in general people have the knack of "smelling out" errors
- There are no rules for error guessing
- The tester is encouraged to think of situations in which the software may not be able to cope.

Copyright © Capgemini 2010. All Rights Reserved.

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.

## 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 2010. All Rights Reserved.

19

Trainer Notes

## 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 2010. All Rights Reserved.

Trainer Notes

## 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 2010. All Rights Reserved.

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

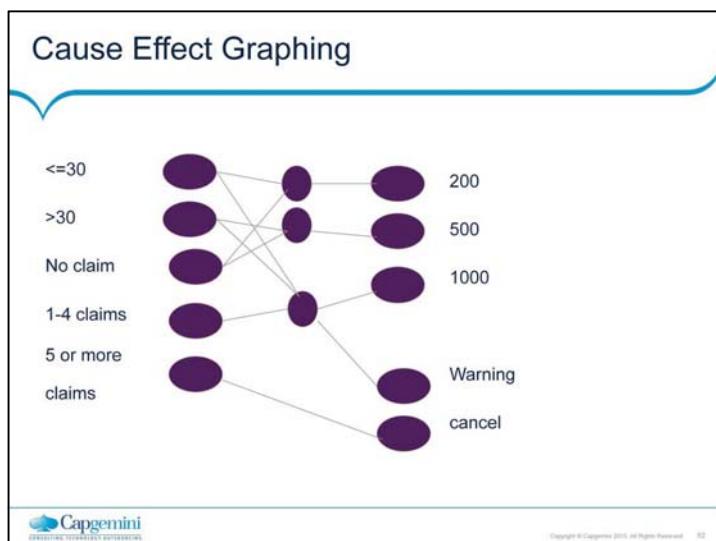
## Cause Effect Graphing

- 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 2010. All Rights Reserved.

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



The above graph covers all the combinations of the scenarios where  
If age $\leq 30$  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

## Cause Effect Graphing

### ▪ Insurance Renewal Decision Table

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

Copyright © Capgemini 2010. 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.

If age<=30 and no claim made premium increase will be 200 else 500. No warning will be sent and no cancellation is done.

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

If 5 or more claims made then cancel policy

## Cause Effect Graphing

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

 Capgemini  
CONSULTING INNOVATION CONSULTANCY

Copyright © Capgemini 2010. 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.

## State Transition Testing

- A testing techniques that aids to validate various states when a program moves from one visible state to another
- It is a techniques in which test cases are designed to execute valid and invalid state transition
  
- 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 2010. All Rights Reserved.

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.

## 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 2012. All Rights Reserved.

### Example :

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

Initially Stack is in Empty state.

As soon as, some elements get added to it with push() method, its state changes to Loaded.

When the element reaches to the Stack's maximum capacity, its state changes from Loaded to FULL.

Similarly, while removing or accessing from the Stack with pop() method, it gets the element into Loaded state. It extracts the topmost element.

## Static vs. Dynamic Testing

### Static Testing

- It is the process of confirming whether the software meets its requirement specification
- Examples : Inspections, walkthroughs and reviews
- It is the process of inspecting without executing on computer
- It is conducted to prevent defects
- It can be done before compilation

### Dynamic Testing

- It is the process of confirming whether the software meets user requirements.
- Examples : structural testing, black-box testing, integration testing, acceptance testing
- It is the process of testing by executing on computer
- It is conducted to correct the defects
- It takes place only after compilation and linking



Copyright © Capgemini 2010. All Rights Reserved.

## Introduction – Test Case Construction & Test Data Preparation

- 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.”



Copyright © Capgemini 2010. All Rights Reserved. 80

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.

## Test Case

- Test Case is 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.
- In other words, Test Case is a planned sequence of actions (with the objective of finding errors)
- Test cases may be designed based on -
  - Values – Valid/Invalid/Boundary/Negative
  - Test conditions
- Test case will be complex if there is more than one expected result.

Copyright © Capgemini 2010. All Rights Reserved.69

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.

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 Case Terminologies

- Pre Condition

- Environmental and state which must be fulfilled before the component/unit can be executed with a particular input value.

- Test Analysis

- is a process for deriving test information by viewing the Test Basis
  - For testing, test basis is used to derive what could be tested

- Test basis includes whatever the test are based on such as System Requirement

- A Technical specification
  - The code itself (for structural testing)
  - A business process

- Test Condition

- 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 2010. All Rights Reserved.

15

## Test Case Terminologies (cont.)

- **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
  - This clearly states what needs to be tested
  - One test condition will have multiple test cases
- **Test Procedure (Test Steps)**
  - A detailed description of steps to execute the test
- **Test Data/Input**
  - Inputs & its combinations/variables used
- **Expected Output**
  - This is the expected output for any test case or any scenario
- **Actual Output**
  - This is the actual result which occurs after executing the test case
- **Test Result/Status**
  - 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 2010. All Rights Reserved.

## 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 2010. All Rights Reserved.

19

**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

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

Copyright © Capgemini 2010. All Rights Reserved. 19

### Features of a good Test Case:

Detecting 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.

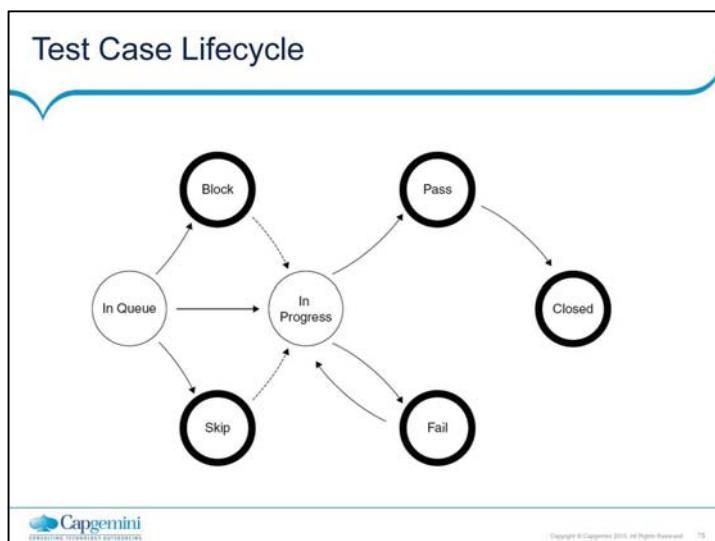
Assess conformance to specification. Any claim made in the specification is checked. Program characteristics not addressed in the specification are not (as part of this objective) checked.

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).

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.

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.



## Test Case Design Techniques

- As Exhaustive testing is impractical, test case design techniques will help us to select test cases more intelligently
- What is a Test Design Technique?
  - A procedure for selecting or designing tests
  - Based on a structural or functional model of the software
  - Successful at finding faults
  - Best' practice
  - A way of deriving good test cases
  - way of objectively measuring a test effort
- Advantages of Test Design Techniques:
  - Different people: similar probability find faults
  - Effective testing: find more faults
  - Efficient testing: find faults with less effort



Copyright © Capgemini 2010. All Rights Reserved.

A testing technique helps us select a good set of tests from the total number of all possible tests for a given system.

Each technique provides a set of rules or guidelines for the tester to follow in identifying test conditions and test cases.

## Test Case Design Techniques (Cont.)

- Test cases are designed based on the following techniques
  - Static (non - execution)
    - Examination of documentation, source code listings, etc.
  - 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, error guessing



Copyright © Capgemini 2010. All Rights Reserved. 19

### Experience based Technique

In experience-based techniques, people's knowledge, skills and background are of prime importance to the test conditions and test cases.

The experience of both technical and business people is required, as they bring different perspectives to the test analysis and design process. Because of the previous experience with similar systems, they may have an idea as what could go wrong, which is very useful for testing.

This technique is used for low-risk systems.

### Exploratory testing

It is a hands-on approach in which testers are involved in minimum planning and maximum test execution.

The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts.

## 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 2010. All Rights Reserved.

19

## Properties of Good Test Data

- Realistic – accurate in context of real life
  - E.g. Age of a student giving graduation exam is at least 18
- Practically valid – data related to business logic
  - E.g. Age of a student giving graduation exam is at least 18 says that 60 years is also valid input but practically the age of a graduate student cannot be 60
- Cover varied scenarios
  - E.g. Don't just consider the scenario of only regular students but also consider the irregular students, also the students who are giving a re-attempt, etc.
- Exceptional data
  - E.g. There may be few students who are physically handicapped must also be considered for attempting the exam



Copyright © Capgemini 2010. All Rights Reserved.

19

## 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 2010. All Rights Reserved. 65

Self explanatory

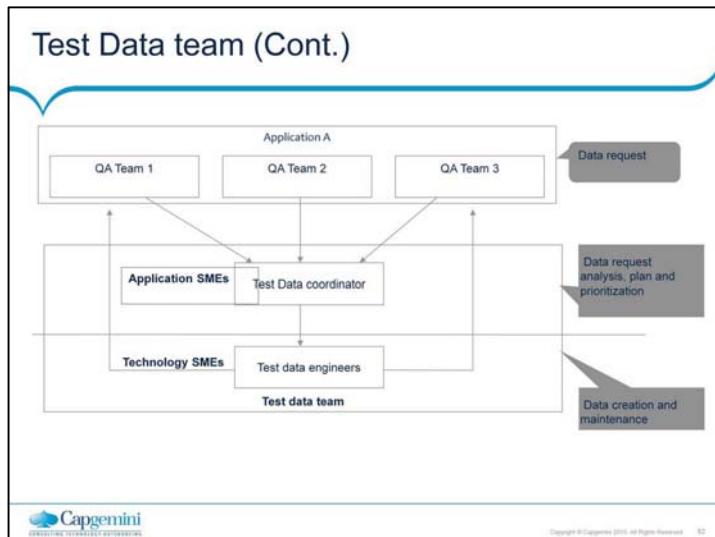
## Test Data team (Cont.)

- 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

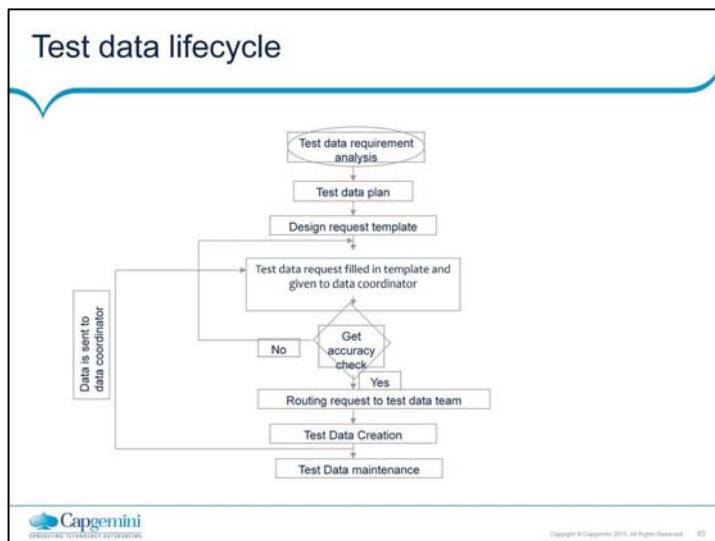


Copyright © Capgemini 2010. All Rights Reserved.

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:

Requirement  
Plan  
Creation  
Usage and maintenance

## 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
  - Parameter-zing 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 2010. All Rights Reserved. 50

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.

## Requirement and Planning (Cont.)

- 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 2016. All Rights Reserved.

85

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 needs 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.

## 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 2010. All Rights Reserved.

## Request Process (Cont.)

### ▪ 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 2010. All Rights Reserved.

Give example of existing project.

## 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 exhibited 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 2012. 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

## Test Data Creation Techniques (Cont.)

- 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



Copyright © Capgemini 2010. All Rights Reserved.

## 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 2010. All Rights Reserved.

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.

## 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



Copyright © Capgemini 2010. All Rights Reserved.

## Test Data Life Cycle - Maintenance (Cont.)

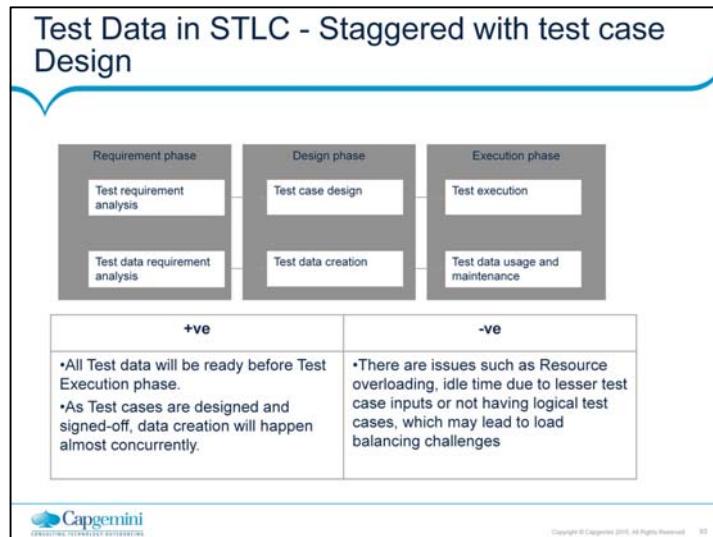
- Other data maintenance activates include:
  - Replacing non-reusable data for regression testing
  - Data Manipulation
  - Responding to change - database schema, code, requirements
  - New test requirements
- **BEST PRACTICES**
  - Data requirement analysis as a part of preparation activity
  - Parameterize 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



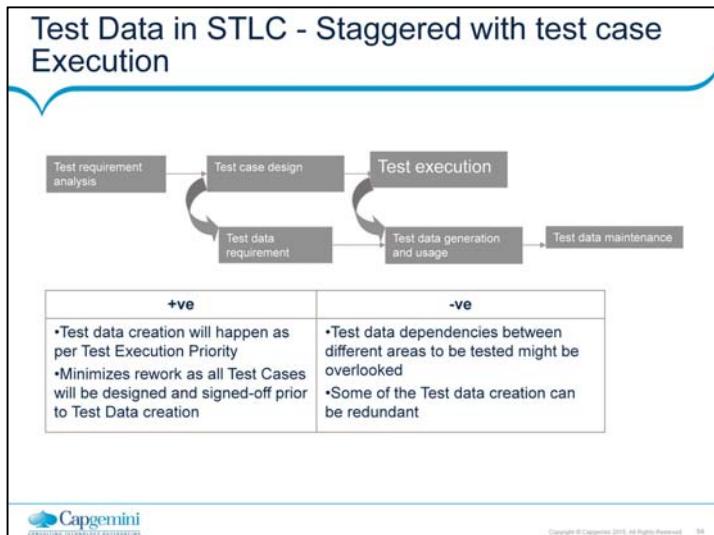
Copyright © Capgemini 2010. All Rights Reserved.

**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.

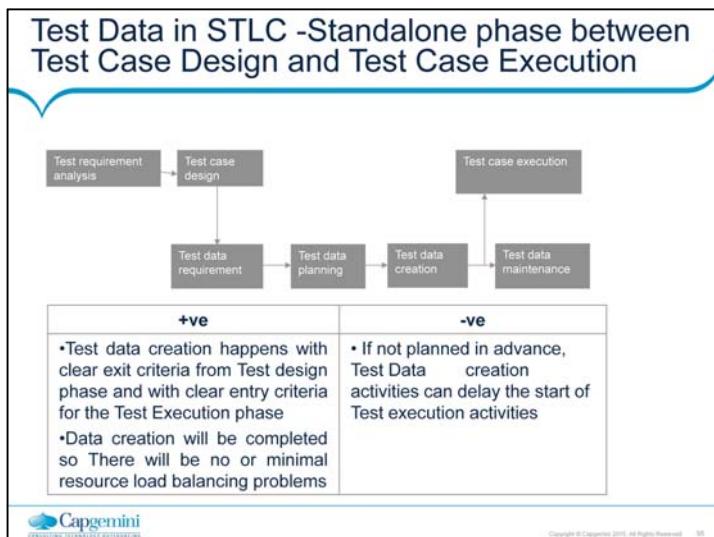
**Data-Non-Reusable:** A test case or a test script that changes the nature of the test data that has been mapped to it comes under this category. An example would be a case where a customer is marked as deceased. Once data is modified after the execution of the test script, the same data cannot be reused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



Self explanatory



Self explanatory



Self explanatory

## What is Positive Testing?

- Positive testing can be performed on the system by entering the valid data as input
  - 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" or "Happy path testing"
  - It is generally the first form of testing that a tester performs on an application
- Example : 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.

Age:  
Testing)

19

Enter only integer values > 18 (Positive



Copyright © Capgemini 2010. All Rights Reserved.

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

## 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 ensures that the business use case is validated

- Limitations of Positive testing:

- Positive tests check for only valid set of values



Copyright © Capgemini 2010. All Rights Reserved.

## What is negative testing?

- The purpose of Negative testing is to break the system and to verify the application response for invalid inputs
  - 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
  - Testing aimed at showing software does not work. Also known as "test to fail"
- 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.

Age:

ABC223

Enter only integer values > 18



Copyright © Capgemini 2010. All Rights Reserved.

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

## 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 2010. All Rights Reserved.

## Positive & Negative 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 up to 15 characters
  - Password textbox should accept any value in between 5-15 chars length
  - Password textbox should accept all numeric & alphabets values
- 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 2010. All Rights Reserved. 100

## 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



Copyright © Capgemini 2010. All Rights Reserved. 103

### 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.



Copyright © Capgemini 2010. All Rights Reserved. 102

## What is Alternate test?

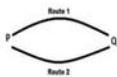
- Sometimes there maybe more than one way of performing a particular function or task with an intent to give the end user more flexibility or for general product consistency
- This is called alternate testing
- Alternate test is a kind of positive testing
- In alternate path testing the test is again 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 2010. All Rights Reserved. 103

### 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.

### 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 2010. All Rights Reserved. 8/19

## 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



Copyright © Capgemini 2016. All Rights Reserved. 109

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 way, you can reuse the same test cases even if the flow changes, merely re-tying them.

## 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



Copyright © Capgemini 2010. All Rights Reserved. 107

## 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



Copyright © Capgemini 2010. All Rights Reserved. 109

## 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?
- Question 5: Incorrect form of logic coverage is :
  - Statement coverage
  - Pole coverage
  - Condition coverage
  - Path coverage



## Review Question

- Question 6: One test condition will have \_\_\_\_\_ test cases.
- Question 7: For Agile development model conventional testing approach is followed.
  - Option: True / False
- Question 8: A test case is a set of \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ developed for a particular objective.
- Question 9: An input field takes the year of birth between 1900 and 2004. State the boundary values for testing this field.
  - 0, 1900, 2004, 2005
  - 1900, 2004
  - 1899, 1900, 2004, 2005
  - 1899, 1900, 1901, 2003, 2004, 2005

Copyright © Capgemini 2010. All Rights Reserved. 119

### 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**

Lesson 3: Testing throughout  
the Software Life Cycle

## Lesson Objectives

- To understand the following topics:
  - Testing throughout the Software Life Cycle
  - Introduction of SDLC and V-Model
  - SDLC and V-Model
  - Iterative Life Cycles
  - Rapid Application Development
  - Rational Unified Process (RUP) Phases
  - RUP Phases and Disciplines
  - Agile Development – Extreme Programming (XP)
  - Testing Phases
  - Introduction of Component Testing
  - Component /Unit Testing



## Lesson Objectives

- To understand the following topics:
  - Introduction of Integration testing
  - Why Integration Testing is Required?
  - Types of Integration testing
  - Top Down Integration Testing
  - Bottom Up Integration Testing
  - Top Down vs. Bottom Up Testing
  - Introduction to System Testing
  - Types of System Testing



## Testing throughout the Software Life Cycle

- Testing is not a stand-alone activity
- It has its place within a SDLC model
- In any SDLC model, a part of testing is focused on Verification and a part is focused on Validation
  - Verification: Is the deliverable built according to the specification?
  - Validation: Is the deliverable fit for purpose?
- Various SDLC models are :
  - V-Model
  - Iterative life cycles
    - Rapid Application Development (RAD)
    - Rational Unified Process (RUP)
    - Dynamic System Development Methodology [DSDM]
    - Agile - Extreme Programming (XP)

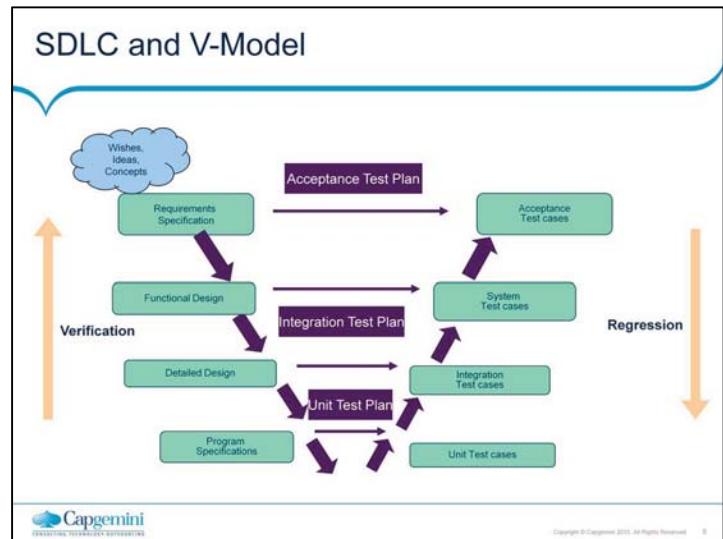
Copyright © Capgemini 2010. All Rights Reserved.

## Introduction of SDLC and V-Model

- 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 development model and V- model of testing
- The V proceeds from left to right, depicting the basic sequence of development and testing activities
- The V 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.



Copyright © Capgemini 2010. All Rights Reserved. 5



#### V Model

Left side shows activities apart from testing

Right side shows Testing activities

Specific Testing activities are carried in parallel to development activities

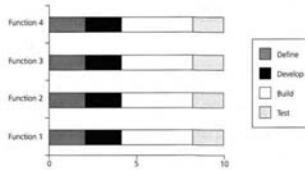
## Iterative Life Cycles

- Iterative life cycle can give early market presence with critical functionality .
- The delivery is divided into increments. Each increment adds new functionality.
- Testing of the new functionality, regression testing, and integration testing are prominent test types performed in iterative life cycle models.
- Simpler to manage because the workload is divided into smaller pieces
- Iterative models are also known as incremental development models
- Examples are
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)
  - Agile development

Copyright © Capgemini 2010. All Rights Reserved

## Rapid Application Development

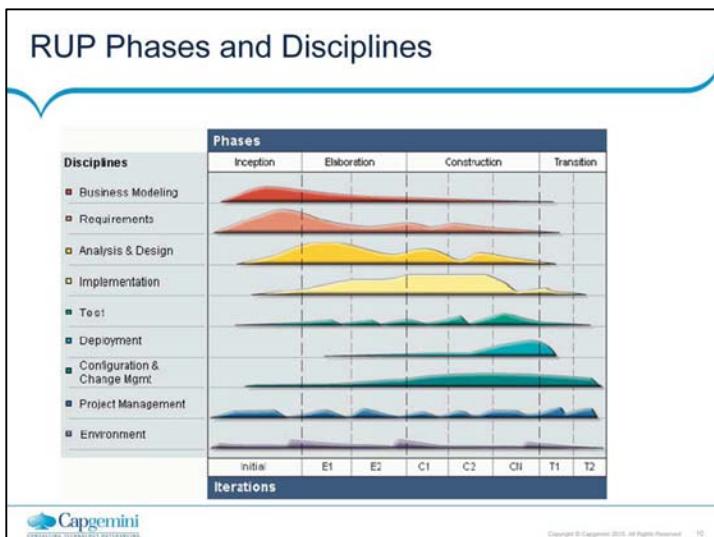
- Components are developed in parallel
- The developments are time-boxed, delivered, and then assembled into a working prototype
- Very quickly gives the customer something to see and use
- Rapid development and rapid response to changing requirements is possible
- Allows early validation of technology risks

Copyright © Capgemini 2010. All Rights Reserved.

## Rational Unified Process (RUP) Phases

- RUP is created by the Rational Software Corporation consisting of four phases and nine disciplines
- Inception
  - The objective is to define scope the system. The business case which includes business context, success factors and financial forecast is established.
- Elaboration
  - The objective is to mitigate the key risk items. The problem domain analysis is made and the architecture of the project gets its basic form.
- Construction
  - The objective is to build the software system. The main focus is on the development of components of the system.
- Transition
  - The objective is to 'transit' the system from development into production. Activities include train the end users and beta testing the system.

Copyright © Capgemini 2010. All Rights Reserved.



### Nine Disciplines

The nine disciplines are performed iteratively throughout the four phases

1. Business Modeling– To understand the business of the organization, possibility of re-engineering business process is explored and potential strategies are evaluated.
2. Requirements - The scope of the project is defined and specification documents for functional and non-functional requirements are prepared.
3. Analysis and Design – The requirements are analyzed and architecture design of the system is made.
4. Implementation – The program source code is developed and unit testing is done.
5. Test - This discipline ensures quality of the system developed. It consists of finding bugs, ensuring that the system works as per the design of the system and meets all requirements mentioned in the specification documents.
6. Deployment - Includes planning and executing delivery of software and supporting documentations ready to be deployed and making the system available to end users.
7. Configuration and Change Management – This includes managing baselines of the project, accepting and managing change requirements, changing and delivering configuration items and managing releases.
8. Project management - This includes assigning tasks, managing risks, tracking progress etc. to ensure on time and within budget delivery of the product.
9. Environment – This includes ensuring proper tools are available whenever required.

## Agile Development – Extreme Programming (XP)

- XP is one of the most well-known agile development life cycle models.
- Some characteristics of XP are:
  - It promotes the generation of business stories to define the functionality
  - It demands an on-site customer for continual feedback and to define and carry out functional acceptance testing
  - It promotes pair programming and shared code ownership amongst the developers
  - It states that component test scripts shall be written before the code is written and those tests are automated
  - It states that integration and testing of the code shall happen several times a day
  - It focuses on implementing the simplest solution to meet today's problems

Copyright © Capgemini 2010. All Rights Reserved. 11

## Testing Phases

- Unit (Component) 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.



Copyright © Capgemini 2014. All Rights Reserved. 10

## Introduction of Component Testing

- The most 'micro' scale of testing to test particular functions, procedures or code modules is called Component testing ; Also called as Module or Unit 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



Copyright © Capgemini 2010. All Rights Reserved. 11

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.

## Component /Unit Testing

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

↓

**Unit**

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

Copyright © Capgemini 2010. All Rights Reserved. 14

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.

## Introduction of Integration testing

- 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

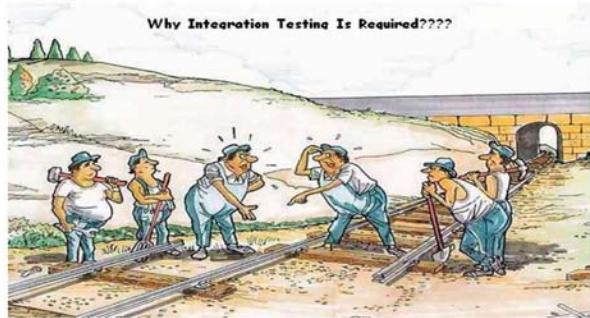


Copyright © Capgemini 2010. All Rights Reserved. 10

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.

## Why Integration Testing is Required?



## Types of Integration testing

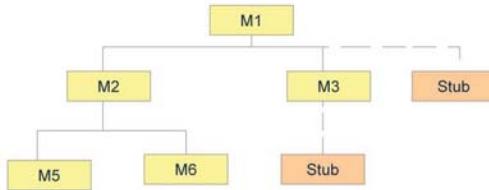
- Modules are integrated by two ways.
  - Non-incremental Testing (Big Bang Testing)
  - Each Module is tested independently and at the end, all modules are combined to form a application
- Incremental Module Testing.
  - There are two types by which incremental module testing is achieved.
- Top down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

Copyright © Capgemini 2010. All Rights Reserved. 17

## 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.



Copyright © Capgemini 2010. All Rights Reserved. 10

#### 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

## Top Down Integration Testing (Cont.)

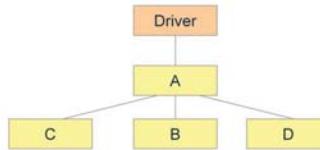
- Integration approach can be done Depth first or Breadth-first.
- Top down testing
  - The main control module is used as a test driver
  - Stubs are substituted for all components directly subordinate to the main control module
  - Depending on the approach subordinate stubs are replaced by actual components

Copyright © Capgemini 2010. All Rights Reserved. 10

## 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



## Bottom Up Integration Testing (Cont.)

- 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

Copyright © Capgemini 2010. All Rights Reserved. 21

## 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.

Copyright © Capgemini 2009. All Rights Reserved.

## 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	

Copyright © Capgemini 2010. All Rights Reserved.

## Introduction to System Testing

- 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.

Copyright © Capgemini 2010. All Rights Reserved. 24

## Types of System Testing

### ▪ Types of System Testing

- Functional Testing
- Performance Testing
- Volume Testing
- Load Testing
- Stress Testing
- Security Testing
- Web Security Testing
- Localization Testing
- Usability Testing
- Recovery Testing
- Documentation Testing
- Configuration Testing
- Installation Testing
- User Acceptance Testing
- Testing related to Changes : Re-Testing and Regression Testing
- Re-testing (Confirmation Testing)
- Regression Testing
- Exploratory Testing
- Maintenance Testing

Copyright © Capgemini 2010. All Rights Reserved. 23

## 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)
- Testing functionality can be done from two perspectives :
  - Business-process-based testing uses knowledge of the business processes
  - Requirements-based testing uses a specification of the functional requirements for the system as the basis for designing tests

Copyright © Capgemini 2010. All Rights Reserved. 20

## 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

Copyright © Capgemini 2010. All Rights Reserved. 27

## Performance Testing (Cont.)

- 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.

Copyright © Capgemini 2010. All Rights Reserved. 20

## 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.



Copyright © Capgemini 2014. All Rights Reserved. 30

## Load Testing

- Volume testing creates a real-life end user pressure for the target software. This tests how the software acts when numerous end users access it concurrently. For e.g.
  - Downloading a sequence of huge files from the web
  - Giving lots of work to a printer in a line



Copyright © Capgemini 2014. All Rights Reserved. 30

## 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.

Copyright © Capgemini 2014. All Rights Reserved. 31

## Stress Testing(Cont.)

- 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

Copyright © Capgemini 2014. All Rights Reserved.

## 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

Copyright © Capgemini 2010. All Rights Reserved. 33

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)

## 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.



Copyright © Capgemini 2014. All Rights Reserved. 50

### 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.,

## 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

Copyright © Capgemini 2010. All Rights Reserved. 33

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.

## 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



Copyright © Capgemini 2010. All Rights Reserved. 30

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?

## Usability Testing (Cont.)

- 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

Copyright © Capgemini 2010. All Rights Reserved. 31

Trainer notes

## 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

Copyright © Capgemini 2010. All Rights Reserved. 30

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.

## 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

Copyright © Capgemini 2010. All Rights Reserved. 30

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.

## 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

Copyright © Capgemini 2010. 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.

## 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

Copyright © Capgemini 2010. All Rights Reserved. A1

## User Acceptance Testing

- A test executed by the end user(s) in an environment simulating the operational environment to the greatest possible extent, that should demonstrate that the developed system meets the functional and quality requirements
- 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 :
  - User Acceptance Testing
  - Operational Acceptance Test / Production Acceptance Test
  - Contract Acceptance Testing
  - Compliance acceptance testing / Regulation Acceptance Testing
  - Alpha Testing
  - Beta Testing

Copyright © Capgemini 2010. All Rights Reserved. 43

User Acceptance Testing (UAT) - Focuses mainly on the functionality thereby validating the fitness-for-use and is performed by the users and application managers

Operational Acceptance Test / Production Acceptance Test - Validates whether the system meets the requirements for operation and may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities

Contract Acceptance Testing - Is performed against a contract's acceptance criteria for producing custom-developed software

Compliance acceptance testing / Regulation Acceptance Testing - Is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

Alpha Testing – It is performed at the developer's site by a cross-section of potential users and members of the developer's organization

Beta Testing - It is performed by a cross-section of users who install it and use it under real-world working conditions

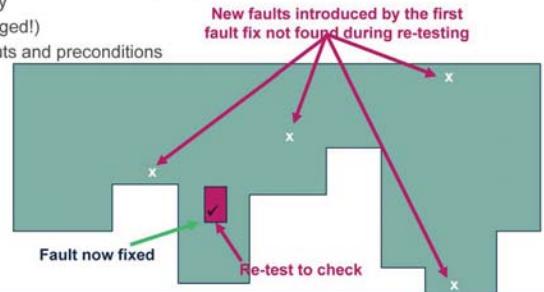
## Testing related to Changes : Re-Testing and Regression Testing

- The target of testing is the testing of changes. It includes below Test Types ;
  - Confirmation testing (Re-testing) – Re-execution of the test after defects are fixed. The test is executed in exactly the same way as it was the first time
  - Regression testing – Ensures that the software is not adversely affected by the changes and critical functionality of the software is still intact. Generally there will be a regression test suite or regression test pack and executed whenever the software changes

Copyright © Capgemini 2014. All Rights Reserved. 43

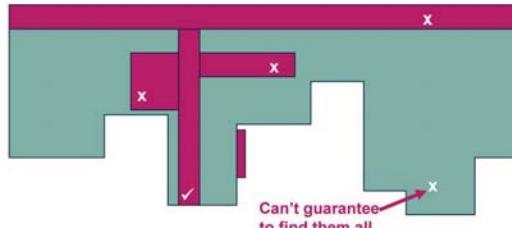
## Re-testing (Confirmation Testing)

- Re-Running failed tests:
  - Must be exactly repeatable
  - Same environment, versions (except for the software which has been intentionally changed!)
  - Same inputs and preconditions



## Regression Testing

- Regression tests are performed:
  - To look for any unexpected side-effects
  - After software changes, including faults fixed
  - When the environment changes, even if application stays same
  - For emergency fixes (possibly a subset)



## Exploratory Testing

- 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



Copyright © Capgemini 2014. All Rights Reserved. 40

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.

## Exploratory Testing - Tips

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



Copyright © Capgemini 2014. All Rights Reserved.

**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.

**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.

**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.

**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.

**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.

## Maintenance Testing

- Testing done after the system is deployed or on existing system is called Maintenance testing
- It is different from maintainability testing, which defines how easy it is to maintain the system
- It consists of two parts:
  1. Testing the changes and defects
  2. Regression tests
- Impact and risk analysis is important activity performed to determine Test efforts
- It is triggered by planned modifications, Ad-hoc corrective modifications, migration, or retirement of the system

Copyright © Capgemini 2014. All Rights Reserved. 60

## 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.



Summary



Copyright © Capgemini 2010. All Rights Reserved. 40

## Review Question

- Question 1: \_\_\_\_\_ is a Quality improvement process
- Question 2: To test a function, the programmer has to write a \_\_\_\_\_, which calls the function to be tested and passes it test data
- Question 3: Volume tests executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
  - True/False
- Question 4: Acceptance testing is not a responsibility of the Developing Organization
  - True/False
- Question 5: Difference between re-testing & regression testing is :
  - re-testing is running a test again; regression testing looks for unexpected side effects
  - re-testing looks for unexpected side effects; regression testing is repeating

Copyright © Capgemini 2010. All Rights Reserved.

### 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

Lesson 4: Test Management &  
Test Case Execution

## Lesson Objectives

- To understand the following topics
  - Test Planning
  - Test Plan Contents (IEEE 829)
  - Test Planning Activities
  - Entry Criteria for Functional Testing
  - Test Case Execution - Pre-execution activities
  - Types of Test Environment
  - Before starting Execution
  - Test Case Execution
  - Exit Criteria for Functional Testing
  - Test Estimation Techniques
  - Factors affecting Test Effort
  - Independent Testing
  - Roles & Responsibilities - Working as Test Leader
  - Roles & Responsibilities - Working as a Tester

Copyright © Capgemini 2010. All Rights Reserved.

## Test Planning

- The Purpose and Substance of Test Plan
  - The Test plan guides our thinking and forces us to confront the challenges that await us
  - The Test plan itself serve as vehicles for communicating with project team members
  - The Test plan helps us manage changes
  - The Test plan becomes a record of previous discussions and agreements between the testers and the rest of the project team
  - There can be different Test plans for different Test levels

Copyright © Capgemini 2010. All Rights Reserved

Test Plan: A document Describing the Scope, approach, resources and schedule of the intended test activities. It identifies, amongst other, test items, the features to be tested, the testing task, the degree of tester independence, the test environment, the test design technique, the entry exit criteria to be used and the rationale for their choice and any risk requiring contingency planning. it is record of the test planning process.

Test Level: A group of test activities that are organized and managed together. A test level is linked to the responsibilities in a project.

## Test Plan Contents (IEEE 829)

- |   |                                      |
|---|--------------------------------------|
| 1. Test Plan Identifier                             | 11. Test Deliverables                |
| 2. References                                       | 12. Testing Tasks                    |
| 3. Introduction                                     | 13. Environmental Needs              |
| 4. Test Items                                       | 14. Staffing and Training Needs      |
| 5. Software Risk Issues                             | 15. Responsibilities                 |
| 6. Features to be Tested                            | 16. Schedule                         |
| 7. Features not to be Tested                        | 17. Planning Risks and Contingencies |
| 8. Test Approach (Strategy)                         | 18. Approvals                        |
| 9. Item Pass/Fail Criteria                          | 19. Glossary                         |
| 10. Suspension Criteria and Resumption Requirements |                                      |

Copyright © Capgemini 2010. All Rights Reserved.

## Test Planning Activities

- Determining the scope, risks and identifying the objectives of testing
- Defining the overall approach of testing, definition of the test levels, entry and exit criteria
- Integrating and coordinating the testing activities into the SDLC activities
- Making decisions about what to test, what roles will perform the test activities, how the test activities should be done, and how the test results will be evaluated
- Scheduling test analysis and design activities
- Scheduling test implementation, execution and evaluation
- Assigning resources for the different activities
- Defining the amount, level of detail and templates for the test documentation
- Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues

Copyright © Capgemini 2010. All Rights Reserved.

## Entry Criteria for Functional Testing

### Functional/System Testing Entry Criteria

- Integration Testing is complete and sign-off is received by Project team
- Integration test results are provided to the QA team within the Integration Execution & Signoff artefact.
- Development team provides a demonstration of application changes prior to promotion to QA Environment
- Code is delivered and successfully promoted to the Functional/System Test Environment as described in Master Test Plan
- Functional/System Test planning is detailed, reviewed and approved within the Master Test Plan

Copyright © Capgemini 2010. All Rights Reserved.

## Entry Criteria for Functional Testing (Cont.)

### 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



Copyright © Capgemini 2010. All Rights Reserved.

## 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

Copyright © Capgemini 2010. All Rights Reserved.

## Test Case Execution - Pre-execution activities(cont.)

- 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 2014. All Rights Reserved.

## Types of Test Environment

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



Copyright © Capgemini 2014. All Rights Reserved. 10

## 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

Copyright © Capgemini 2014. All Rights Reserved. 11

## Test Case 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)
- 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)

Copyright © Capgemini 2014. All Rights Reserved. 10

## Test Case Execution (Cont.)

- 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 2014. All Rights Reserved. 13

## 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

Copyright © Capgemini 2014. All Rights Reserved. 10

## Exit Criteria for Functional Testing (Cont.)

### 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 2014. All Rights Reserved. 10

## Test Estimation Techniques

- Estimating the efforts required for testing is one of the major and important tasks in SDLC
- Correct estimation helps in testing the software with maximum coverage
- Software Testing Estimation Techniques :
  - Work breakdown structure (WBS) – Breaking down the large activities and tasks into smaller, more manageable tasks.
  - Bottom-up estimation - Estimate for efforts, duration, dependencies etc for lowest level tasks from WBS and roll-up to arrive total estimate
  - Top-down estimation - Deriving estimates from similar projects
  - Parametric technique – Estimating based on some parameters. E.g. average effort per test case

Copyright © Capgemini 2014. All Rights Reserved. 10

## Factors affecting Test Effort

- Characteristics of the product:
  - The quality of the specification
  - The size of the product
  - The complexity of the problem domain
- Characteristics of the development process:
  - The stability of the organization
  - Tools used
  - Test process
  - Skills of the team members
  - Time pressure
- The outcome of testing:
  - The number of defects
  - The amount of rework required

Copyright © Capgemini 2014. All Rights Reserved. 17

## Independent Testing

- Independent testing is the degree of independence to which testing is performed.
- Levels of Test Independence:
  - Independent testers within the development teams.
  - Independent test team or group within the organization, reporting to project management or executive management.
  - Independent testers from the business organization, user community and IT.
  - Independent test specialists for specific test targets such as usability testers, security testers or certification testers
  - Independent testers outsourced or external to the organization

Copyright © Capgemini 2010. All Rights Reserved. 10

## Independent Testing (Cont.)

### ■ Advantages

- Independent testers uncover different defects, and are unbiased
- An independent tester can verify assumptions made during specification and implementation of the system
- Usually a Cost saving
- Better skills, more effective testing and fewer defects getting into production

### ■ Drawbacks

- Isolation from the development team (if treated as totally independent).
- Can be the bottleneck as the last checkpoint
- Developers lose a sense of responsibility for quality
- Can be a greater cost – need to consider viability
- For Third Party test outsourcing, the project carries the risk

Copyright © Capgemini 2010. All Rights Reserved. 10

## Roles & Responsibilities - Working as Test Leader

- Strategy & Management
  - Write and review the test strategy
  - Plan testing effort – context, risks & approach
  - Proactive representation in project activities – ensure testing has correct focus
  - Ensure proper configuration management of Testware
  - Determine what should be automated and select most appropriate Test Tools
  - Management and definition of the test environmental requirements
  - Define the test schedule based on the delivery of code in to test
- Monitor
  - Define, record and continually review the testing project metrics
  - Monitor test progress against the test schedule
  - Write the test summary reports
- Control
  - Adapt testing effort based results and progress

Copyright © Capgemini 2014. All Rights Reserved.

Test Leader: The Person responsible for the management of testing activities and resources. The individual who directs control, administers, plan and regulate the evaluation of a test object.

## Roles & Responsibilities - Working as a Tester

- Review and contribute to test plans
- Analyze, review and assess user requirements, specifications and models for testability
- Create, review Test specifications
- Set up the test environment
- Define, prepare and acquire test data
- Implement tests on all test levels, execute and log the tests
- Evaluate the results and document the deviations (defects/issues) from expected results
- Use various test tools as required
- Automate tests

Copyright © Capgemini 2014. All Rights Reserved. 21

Tester: A skilled Professional who is involved in the testing of component or system

## Summary

■ In this lesson, you have learnt:

- Test management is covered from a skills perspective, focusing on test execution and defect reporting and handling.
- Managing the Test Activities
- Role of test leader and testers
- Test planning activities
- Template of Test document artifacts such as test plan and test case designs
- Entry and exit criteria of tests
- Test execution activities



Copyright © Capgemini 2010. All Rights Reserved. 22

## Review - Questions

- Question 1: The degree of independence to which testing is performed is known as \_\_\_\_\_.
- Question 2: there can be different test plans for different test levels
  - Option: True / False
- Question 3 Exit criteria are used to report against and to plan when to begin testing
  - Option: True / False
- Question 4: Which of the following are Test Environments
  - Unit Test Environment
  - QA Test Environment
  - Simple Test Environment
  - Product Environment

Copyright © Capgemini 2010. All Rights Reserved. 21

## Review – Match the Following

1. Entry Criteria

2. Exit Criteria

3. Test Approach

4. Failure based testing

A. Consultative approach

B. Acceptance criteria

C. Methodical approach

D. Completion criteria



Testing Concepts

Testing Metrics

## **Testing Concepts**

Lesson 5: Testing Metrics

## Lesson Objectives

- To understand the following topics
  - Monitoring the Progress
  - Metrics of Test Progress
  - Reporting Test Status
  - Test Control
  - Configuration Management & Configuration Control
  - Products for Configuration Management in Testing
  - Definition of Metrics
  - Need of Metrics
  - Metrics for Testing
  - Types of Metrics
  - Types of Metrics – Project Metrics
  - Types of Metrics – Process Metrics
  - Types of Metrics – Productivity Metrics
  - Types of Metrics – Closure Metrics

Copyright © Capgemini 2010. All Rights Reserved.

## Monitoring the Progress

### ■ Why Test monitoring is necessary?

- To know the status of the testing project at any given point in time
- To provide visibility on the status of testing to other stake holders
- To be able to measure testing against defined exit criteria
- To be able to assess progress against Planned schedule & Budget

### ■ IEEE 829 Standard: Test Log Template Content

- Test Log identifier
- Description (items being tested, environment in which the testing is conducted)
- Activity and event entries (execution description, procedure results, environmental information, anomalous events, incident report identifiers)

Copyright © Capgemini 2010. All Rights Reserved.

## Metrics of Test Progress

- Metrics should be collected during and at the end of a test level. They are also valuable input into process improvement. Common metrics for test progress monitoring include:
  - The extent of completion of test environment preparation
  - The extent of test coverage achieved, measured against requirements, risks, code, configurations or other areas of interest
  - The status of the testing compared to various test milestones
  - The economics of testing, such as the costs and benefits of continuing test execution in terms of finding the next defect or running the next test.

Copyright © Capgemini 2010. All Rights Reserved.

## Reporting Test Status

- Reporting test status is about effectively communicating our findings to other project stakeholders. It is usually done through Test Summary Report
- IEEE 829 Standard: Test Summary Report Template
  - Test summary report identifier
  - Summary
  - Variances
  - Comprehensive assessment
  - Summary of results
  - Evaluation
  - Summary of activities
  - Approvals

Copyright © Capgemini 2010. All Rights Reserved.

## Test Control

- Test control is the response to Test Monitoring and Test Reporting that allows us to be IN CONTROL of the project
- Issues need to monitored and reported
- The process of control is the corrective actions required to put a testing effort (project) back on track
- For Example:
  - Re-prioritize tests when an identified risk
  - Change the test schedule based on availability of a test environment

Copyright © Capgemini 2010. All Rights Reserved.

## Configuration Management & Configuration Control

- Configuration Management:

- A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item

- Configuration Control or Version control:

- An element of configuration management, consisting of evaluation, coordination, approval or disapproval and implementation of changes to configuration items after formal establishment of their configuration identification

Copyright © Capgemini 2014. All Rights Reserved

## Products for Configuration Management in Testing

- Test plans
  - Test designs
  - Test cases:
    - Test input
    - Test data
    - Test scripts
    - Expected results
  - Actual results
  - Test tools
- 
- What would not be under configuration management?
    - Live data

Copyright © Capgemini 2014. All Rights Reserved.

## 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.



Copyright © Capgemini 2010. All Rights Reserved.

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  
Functionality  
Schedule  
Cost

## Need 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



Copyright © Capgemini 2010. All Rights Reserved. 10

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

## 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 =  $100 * \text{No. of pre-delivery defects} / \text{Total No. of Defects}$


Copyright © Capgemini 2010. All Rights Reserved. 11

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

Defects are measured as found in  
 Reviews  
 Testing  
 Acceptance  
 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  
 $\text{Defect Age} = 4-2*10/40 = 1$  ----- 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,  
 $\text{DRE} = 100 * 5/20 = 100*0.25=25$

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.

## Metrics for Testing (Cont.)

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



Copyright © Capgemini 2010. All Rights Reserved.

### 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

$RE = 100 * 20 / 40 = 50$ . 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.

$CFDT = \text{Total efforts spent on testing} / \text{defects found in testing} = 60 / 30 = 2$  hrs

## Metrics for Testing (Cont.)

- 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

Copyright © Capgemini 2016. All Rights Reserved. 13

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.

## Metrics for Testing (Cont.)

- 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 \* efforts spent on fixing or reworking the post-delivery defects)


Copyright © Capgemini 2010. All Rights Reserved. 14

**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 \* 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.

## Metrics for Testing (Cont.)

### ■ Test Case Effectiveness

- Test Case Effectiveness = # of defects detected using the test cases \* 100/ total # of defects detected in testing
- This metrics defines the effectiveness 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



Copyright © Capgemini 2016. All Rights Reserved. 10

Test Case Effectiveness (TCE) : This metrics defines the effectiveness 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.

## Metrics for Testing (Cont.)

### ■ 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

Copyright © Capgemini 2010. All Rights Reserved. 10

### 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 =  $30*100/40 = 75$

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.

## Metrics for Testing (Cont.)

- 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



Copyright © Capgemini 2010. All Rights Reserved. 17

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 =  $20/100 = 0.50$

## Metrics for Testing (Cont.)

- 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

Copyright © Capgemini 2010. All Rights Reserved. 10

## Metrics for Testing (Cont.)

- 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



Copyright © Capgemini 2010. All Rights Reserved.

### 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.

## Types of Metrics

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



Copyright © Capgemini 2010. All Rights Reserved. 20

## Types of Metrics – Project Metrics

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



Copyright © Capgemini 2010. All Rights Reserved. 21

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.

## Types of Metrics – 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



Copyright © Capgemini 2010. All Rights Reserved. 22

### 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

## Types of Metrics – Process Metrics (Cont.)

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

Copyright © Capgemini 2010. All Rights Reserved. 11

### 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

## Types of Metrics – Productivity Metrics

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

Copyright © Capgemini 2016. All Rights Reserved. 24

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.

## Types of Metrics – Closure Metrics

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



Copyright © Capgemini 2010. All Rights Reserved. 23

### 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.

## 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

Copyright © Capgemini 2010. All Rights Reserved. 20

## 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 – 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

Tool Supporting for Testing

## **Testing Concepts**

Lesson 6: Tool Supporting for  
Testing

## Lesson Objectives

- To understand the following topics:

- Tool support for Testing
- Test Tools Classification
- Tool Support for Management of Testing and Test
- Tool support for Static Testing
- Tool support for Test Specification
- Tool support for Test Execution & Logging
- Tool support for Performance & Monitoring
- Tool support for specific Testing Needs
- Need of Software Testing Tools
- Potential Benefits of using Tools
- Risks of using Tools
- Special Considerations for some Types of Tools
- Introducing a Tool into an Organization



## Tool support for Testing

- Test tools can be used for activities that support testing:
  - Directly used in testing such as test execution
  - Help in managing the testing process
  - Used for exploration
  - Aids in testing such as spreadsheet
  
- Tool support for testing can have following purposes :
  - Improve the efficiency of test activities by automating or supporting manual test activities
  - Automate activities that cannot be executed manually or require significant resources
  - Increase reliability of testing

Copyright © Capgemini 2014. All Rights Reserved.

### Tool support for Testing

Tools used for test activities in software life cycle are called Computer Aided Software Testing or CAST Tools.

## Test Tools Classification

- Tools can be classified based on following criteria :
  - Purpose
  - Commercial / free / open-source / shareware
  - Technology used
  - Tools that support Testing activities
  - Intrusive Tools - Can affect the actual outcome of the test
    - E.g. The actual timing may be different due to the extra instructions that are executed by the tool. This is called the Probe effect
  - Some tools offer support appropriate for developers
    - E.g. Tools used for component and component integration testing

Copyright © Capgemini 2014. All Rights Reserved.

## Tool Support for Management of Testing and Test

- Test Management Tools

- Provide interfaces for executing tests
- Track defects
- Manage requirements
- Support for quantitative analysis
- Reporting of the test objects
- Tracing the test objects to requirements

- Requirements Management Tools

- Store requirement statements
- Store the attributes for the requirements
- Provide unique identifiers
- Support tracing the requirements to individual tests
- Help to identify inconsistent or missing requirements

Copyright © Capgemini 2014. All Rights Reserved. 1

## Tool Support for Management of Testing and Test(Cont.)

- Incident Management Tools (Defect Tracking Tools)
  - Store and manage incident reports
  - Help in managing the life cycle of incidents, optionally provide support for statistical analysis
- Configuration Management Tools
  - Not strictly test tools but are necessary
  - storing information about versions and builds of the software and testware
  - traceability between software and testware
  - release management, baselining, and access control.

Copyright © Capgemini 2014. All Rights Reserved.

## Tool support for Static Testing

### ■ Review Tools

- Assist with review processes, checklists, review guidelines
- Used to store and communicate review comments
- Report on defects and effort
- Provides aid for online reviews for large or geographically dispersed teams.

### ■ Static Analysis Tools

- Help developers and testers find defects prior to dynamic testing
- Provide support for enforcing coding standards
- Help in planning or risk analysis by providing the metrics for the code (e.g., complexity)

### ■ Modeling Tools

- Used to validate software models (e.g., physical data model for a RDBMS)
- Help in finding defects
- Aid in generating some test cases based on the model



Copyright © Capgemini 2019. All Rights Reserved. 7

Static analysis tools are useful during design and coding phases by identifying gaps in the code.

## Tool support for Test Specification

### ■ Test Design Tools

- Generate test inputs or executable tests
- Generate test oracles from requirements, graphical user interfaces, design models or code

### ■ Test Data Preparation Tools

- Manipulate databases, files or data transmissions
- Set up test data to be used during the execution of tests
- Ensure security through data anonymity



Copyright © Capgemini 2014. All Rights Reserved.

Test design tool supports the test design activity by generating test inputs, from a specification that may be held in a CASE tool repository. For example, if the requirements are kept in a requirements management or test management tool, or in a CASE tool used by developers, then it is possible to identify the input fields, including the range of valid values. If the valid range is stored, the tool can distinguish between values that accept and generates an error message.

## Tool support for Test Execution & Logging

### ■ Test Execution Tools

- Executes tests automatically or semi-automatically using stored inputs and expected outcomes
- Uses a scripting language and usually provides a test log for each test run
- Records tests, supports scripting languages or GUI-based configuration for parameterization and other customization

### ■ Test Harness/Unit Test Framework Tools

- Tests the components or parts of a system by simulating the environment in which that test object will run
- Provision of mock objects as stubs or drivers



Copyright © Capgemini 2019. All Rights Reserved. 8

Test harness and drivers, and dynamic analysis tools are used during integration testing.

## Tool support for Test Execution & Logging (Cont.)

### ■ Test Comparators

- Determine differences between files, databases or test results
- A test comparator may use a test oracle, especially if it is automated

### ■ Coverage Measurement Tools

- Measure the percentage of specific types of code structures that have been exercised through intrusive or non-intrusive means
- E.g., Statements, branches or decisions and module or function calls

### ■ Security Testing Tools

- Evaluates the security characteristics of software
- Evaluates the ability of the software to protect data confidentiality, integrity, authentication, authorization, availability, and non-repudiation
- Focused on a particular technology, platform and purpose



Copyright © Capgemini 2014. All Rights Reserved.

Test run and comparison tools are used in all levels of testing to compare results. Security testing can be started during coding phase and can continue till the system is moved to production, and sometimes after.

## Tool support for Performance & Monitoring

- Dynamic Analysis Tools
  - Find defects only when software is executing, such as time dependencies or memory leaks
  - Used in component and component integration testing and when testing middleware
- Performance Testing/Load Testing/Stress Testing Tools
  - Monitor and report on how a system behaves under a variety of simulated usage conditions
  - The simulation of load is achieved by creating virtual users (VUsers) carrying out a selected set of transactions
- Monitoring Tools
  - Continuously analyze, verify and report on usage of specific system resources
  - Give warnings of possible service problems

Copyright © Capgemini 2014. All Rights Reserved. 11

Performance measurement tools are used in system and acceptance testing to measure the system performance.

Dynamic analysis tool provides runtime information on the state of the software code that includes allocation, use, and de-allocation of resources, and flag of unassigned pointers or pointer arithmetic faults.

## Tool support for specific Testing Needs

### ■ Data Quality Assessment

- Review and verify the data conversion and Migration rules
- Verify data against pre-defined context specific standard
- Other testing tools exist for usability testing



Copyright © Capgemini 2010. All Rights Reserved. 10

## Need of Software Testing Tools

- Reasons for acquiring tools to support testing:
  - Doing certain tasks that are better done by a computer than by a person
  - Ever-shrinking schedule and minimal resources
  - It involves automating a manual process of testing
  - Eliminating human error

Copyright © Capgemini 2010. All Rights Reserved. 10

## Potential Benefits of using Tools

- Reduction of repetitive work
- Greater consistency and repeatability
- Objective assessment
- Ease of access to information about tests or testing

Copyright © Capgemini 2014. All Rights Reserved. 14

Manual testing is dependent on the style and nature of the individual performing the test. Hence, it differs from person to person. Use of tools remove this variation as they can only perform the task they are programmed for.

## Risks of using Tools

- Unrealistic expectations from the tool
- Under estimating the time, cost and effort while initial introduction of a tool
- Under estimating the time and effort needed to achieve significant and continuing benefits from the tool
- Under estimating the effort required to maintain the test assets
- Over-reliance on the tool
- Poor response from vendor for support, upgrades and defect fixes
- Risk of suspension of open-source / free tool project

Copyright © Capgemini 2010. All Rights Reserved.

## Special Considerations for some Types of Tools

### ■ Test Execution Tools

- Often requires significant effort in order to achieve significant benefits.
- Capture/playback does not scale to large numbers of automated test
- Captured script may be unstable when unexpected events occur
- Technical expertise in the scripting languages needed for all approaches
- The expected results for each test need to be stored for later comparison
- Various types of scripting to be considered
  - Linear scripts
  - Structured scripts
  - Shared scripts
  - Data-driven scripts
  - Keyword-driven scripts



Copyright © Capgemini 2019. All Rights Reserved. 10

## Special Considerations for some Types of Tools (Cont.)

- Static analysis tools:

- Can identify potential problems in code before the code is executed
- Can help to check that the code is written to coding standards
- Tools can generate a large number of messages
  - E.g. By finding the same thing after every few lines
- The aim is to produce code that will be easier to maintain in the future
- A filter on the output could eliminate less important messages and highlight more important messages

Copyright © Capgemini 2014. All Rights Reserved. 10

## Special Considerations for some Types of Tools (Cont.)

- Test management tools:
  - These tools can provide a lot of useful information, but the information may not be in the required form
  - Additional work needed to produce interfaces to other tools
  - A defined test process needs to be set before test management tools are introduced
  - If the testing process is working well manually, then a test management tool can help to support the process and make it more efficient



Copyright © Capgemini 2014. All Rights Reserved. 10

## Introducing a Tool into an Organization

- Main Considerations in Selecting a Tool:
  - Assessment of the organization's maturity
  - Identification of the areas where tool support will help to improve testing processes
  - Evaluation of tools against clear requirements and objective criteria
  - Proof-of-concept to see whether the product works as desired
  - Evaluation of the vendor or open-source network of support
  - Identifying and planning internal implementation

Copyright © Capgemini 2019. All Rights Reserved. 10

## Introducing a Tool into an Organization (Cont.)

- Pilot Project:

- One of the ways to do a proof-of-concept is to have a pilot project as the first thing done with a new tool. This will use the tool on a small scale, with sufficient time to explore different ways of using the tool.

- The objectives for a pilot project for a new tool are:

- To learn more about the tool (more detail, more depth)
  - Evaluate how the tool fits with existing processes and practices, and determine scope to change
  - Decide on standard ways of using, managing, storing and maintaining the tools and test assets
  - Assess whether the benefits will be achieved at reasonable cost



Copyright © Capgemini 2019. All Rights Reserved.

## Introducing a Tool into an Organization (Cont.)

- Success factors:
  - Success is not guaranteed or automatic when implementing a testing tool.
- Some of the factors that have contributed to success :
  - Incremental roll-out (after the pilot) to the rest of the organization
  - Adapting and improving processes, testware and tool artifacts
  - Providing adequate training, coaching and mentoring for new users
  - Defining and communicating guidelines for the use of the tool
  - Implementing a continuous improvement mechanism
  - Monitoring the use of the tool, benefits achieved and lessons learned

Copyright © Capgemini 2018. All Rights Reserved. 21

## Summary

- In this lesson, you have learnt:
  - Various types of testing Tools
  - Benefits and Risks of using Tools
  - Introducing Tools into an Organization



Copyright © Capgemini 2018. All Rights Reserved. 

## Review - Questions

- Question 1: The \_\_\_\_\_ tool is used to detect a memory leak.
- Question 2 Tool supported for static testing is a good way to force failures into the software.
  - Option: True / False
- Question 3: Goal of Pilot Project for tool evaluation is to evaluate how the tool fits with existing processes and practices.
  - Option: True / False



# Testing Concepts – V2.0

## 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
16/06/2016	5	Neelima Padmawar, Leena Pangarkar	Post integration material revamp

## Table of Contents

<i>Document Revision History</i> .....	2
<i>Table of Contents</i> .....	3
<i>Lab 1. Software Testing Basics – White Box Testing</i> .....	5
1.1 <i>Create Condition Coverage Matrix</i> .....	5
1.2 <i>Consider the below code to solve the following subsections</i> .....	5
1.2.1 <i>Create condition coverage matrix</i> .....	6
1.2.2 <i>Validate the coding standards</i> .....	6
1.2.3 <i>Write test cases</i> .....	7
1.3 <i>Draw Flow Graph &amp; Determine CC</i> .....	7
<i>Lab 2. Software Testing Basics – Black box Testing</i> .....	8
2.1 <i>Validate Date field</i> .....	8
2.2 <i>Validate Command Line utility</i> .....	8
2.3 <i>Validate Phone Number field</i> .....	8
2.4 <i>Validate Password Field</i> .....	8
2.5 <i>Determine ECP &amp; BVA</i> .....	8
<i>Lab 3. Creating Test Scenario – Post an Ad on EXCHANGE on WEB</i> .....	10
3.1: <i>Test Scenario Case Study. ‘Posting an Ad on EXCHANGE on WEB (EoW)’</i> .....	10
3.2: <i>Steps to follow to post an Ad</i> .....	10
3.3 <i>Rules:</i> .....	13
<i>Lab 4. Creating Test Cases – Customer Complaint Form</i> .....	14
4.1 <i>Case Study : Customer Complaint Form</i> .....	14
4.2 <i>Instructions</i> .....	14
4.3 <i>Rules</i> .....	15
<i>Lab 5. Creating Test Cases – Conference Room Booking</i> .....	16
5.1: <i>Case Study. ‘ONLINE CONFERENCE ROOM BOOKING’ on Intranet</i> .....	16
<i>Booking Instructions</i> .....	16
<i>Making a new booking:</i> .....	16
<i>Viewing / Cancellation of Bookings:</i> .....	17
<i>Invoke https://intranet.igate.com</i> .....	17
<i>Intranet home →Employee Corner →Conference Room Booking</i> .....	18
<i>Conference Room Booking → New Booking</i> .....	18

<i>HOME → Conference Room Booking → New Booking .....</i>	19
<i>If booking is successful, the following screen is displayed.....</i>	19
<i>HOME → Conference Room Booking → New Booking .....</i>	19
<i>The booking status is also shown under My Bookings as shown below:- .....</i>	20
<i>HOME → Conference Room Booking → Booking Status.....</i>	20
<b>Lab 6: Creating Test Cases – Cyber Shopee.....</b>	<b>21</b>
6.1 Case Study: CyberShoppe.....	21
6.2 Process Work Flow: .....	22
6.2.1 Visit CyberShopee Website .....	22
6.2.2 Register.....	22
6.2.3 Admin Module .....	24
6.2.4 Customer Module.....	26
6.2.5 Dealer Module.....	30

## Lab 1. Software Testing Basics – White Box Testing

Goals	<ul style="list-style-type: none"> <li>• Learn to prepare Condition Coverage Matrix</li> <li>• Learn to determine Cyclomatic Complexity</li> </ul>
Time	180 Minutes

### 1.1 Create Condition Coverage Matrix

Table: Template of Condition Coverage Matrix with an Example:

Test Condition	Description	Expected Output
cond1 = true cond2 = notEval	Raining and Wind is true. Therefore, else condition is not evaluated	Stay Inside
cond1 = true cond2 = notEval	Evening is true. Therefore, else condition is not evaluated	Stay Inside
cond3=true	Sunshines is true	Go to beach

See the below specification:

```

IF          (it rains AND there is wind) OR it is evening
THEN        stay inside
ELSE        go outside
END IF
IF          the sun shines
THEN        go to the beach
END IF

```

How many test situations result from applying the coverage type of decision points with multiple condition coverage and how many test situations have “go to the beach” as the result?

### 1.2 Consider the below code to solve the following subsections

```

#include<stdio.h>

void main() {
    A, B, C ;
    PrintF ("Enter three numbers : ");
    scanf("%d %d %d", &A, &B, &C);
    if ((A>B) && (A>C))
    {
        printf ("%d is greater", A);
    }
}

```

```

}
elseif (B>C)
{
    printf ("%d is greater", B)
}
else
{
    printf ("%d is greater", C)
}

```

### 1.2.1 Create condition coverage matrix

cond1 = true	cond2 = not eval	cond3 = false
cond1 = ?	cond2 = ?	cond3 = ?
cond1 = ?	cond2 = ?	cond3 = ?

Table: Template of Condition Coverage Matrix with an Example:

Test Condition	Description	Expected Output
cond1 = true cond2 = notEval cond3 = false	A>B and A>C both are true. Therefore, elseif and else conditions are not evaluated	A is greater

### 1.2.2 Validate the coding standards

Note: You can make use of Code Review Check list explained in Lesson 02. You can also make use of sample template given below

Table: Template of Code Review Checklist

Sr. No.	Question	Remark (Yes / No)
<b>Syntactical Errors</b>		
1	Does every statement has a delimiter?	
2	Are the in-built functions spelled properly?	
<b>Data Declaration Errors</b>		
3	Have all variables been explicitly declared?	
4	Are variables properly initialized in declaration sections?	
<b>Comparison Errors</b>		
5	Are there any comparisons between variables having inconsistent data types?	
<b>Control Flow Errors</b>		
6	Does every cause has an effect?	
<b>Input / Output Errors</b>		
7	All Input statements handled correctly?	
8	All Output statements handled correctly?	

### 1.2.3 Write test cases

Consider the following scenarios to write the test cases for the above code:

- As you see in the above code, the data type of the variables is not specified. Write test case to validate whether the values stored in the variable A, B & C are integer data type
- Write test cases to validate the Comparison code

### 1.3 Draw Flow Graph & Determine CC

Draw corresponding flow graph and find the cyclomatic complexity (CC) for the following specifications:

Specification 1	Specification 2
<pre> IF      A AND B THEN    C=50 ELSE         IF      C AND D         THEN   Error message         ENDIF ENDIF </pre>	<pre> IF      customer no. &gt; 200 AND article group = 330 THEN discount = 5% ENDIF IF      region code = 4 OR 8 THEN invoice type = A ELSE   invoice type = B ENDIF </pre>

## Lab 2. Software Testing Basics – Black box Testing

Goals	<ul style="list-style-type: none"><li>• Learn to apply basic techniques for writing test cases.</li><li>• Learn to prepare finite and best suitable test cases</li></ul>
Time	180 Minutes

### 2.1 Validate Date field

Format of Date field is : dd/mm/yyyy

### 2.2 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
```

### 2.3 Validate Phone Number field.

Format of the number is :

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

### 2.4 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.

### 2.5 Determine ECP & BVA

1. Consider a scenario where a 'Driver on Hire' agency provides most reliable and affordable drivers on hire. The monthly salary structure configured for the drivers of this

agency is : drivers up to 25 years old get a pay of Rs. 15000/- ; drivers of age 65 and older get a pay of 5% more and drivers of age 40 and older get a pay of 10% more than that of the drivers up to 25 years get. How many equivalence classes are distinguished in the above? Also, which values are chosen for making test cases when the normal variant of the boundary values analysis is used?

2. Consider a scenario where a 'Winter Sale' provides heavy discounts. A person he/she of age less than 8 years old (<8), or a person aged between 35 and 45 years (>35 and <45) or older than 60 years (>60) is eligible for the discounts in Winter Sale. How many equivalence classes can be distinguished in this example?

## Lab 3.Creating Test Scenario – Post an Ad on EXCHANGE on WEB

Goals	<ul style="list-style-type: none"><li>• Read 'INSTRUCTIONS' before starting the assignment.</li><li>• Understand the application and develop Test Scenarios</li></ul>
Time	60 minutes

### 3.1: Test Scenario Case Study. 'Posting an Ad on EXCHANGE on WEB (EoW)'

[www.eow.com](http://www.eow.com) is the most preferred online web application in India for exchanging/selling of old products. In this case study, customer posts and advertisement of his product on the EoW portal.

Participants need to write Test Scenarios for testing Buttons, Hyper Links, and Form fields.

**Note:** Trainer will discuss the Test Scenario template and will share the same with participants

### 3.2: Steps to follow to post an Ad

1. The customer visit <https://www.eow.com> site. To login, click on '**Login with phone & OTP**'. Login using Friendsbook account is out of scope.
2. The customer enters 10 digit mobile no and clicks on '**Get OTP**'. The application sends an OTP (One Time Password) on his phone no.
3. The portal asks for the OTP confirmation code. User can enter the code and '**CONFIRM**' to continue or **CANCEL** it. If the user has timed out (at max. 60 sec) in typing the OTP code, he can click **Resend code** for receiving another OTP code.
4. The portal displays the home page. Customer clicks on '**Submit a Free Ad**' to post a new add. Searching for products is out of scope.
5. Form of '**Submit a Free Classified Ad**' is displayed. Fill all the details in the form and click '**Submit**' button
6. The portal asks for the OTP sent on phone no. that is given while submitting the free classified Ad. User can enter the code and '**CONFIRM**' to continue or **CANCEL** it. If the user has timed out (at max. 60 sec) in typing the OTP code, he can click **Resend code** for receiving another OTP code.
  - On entering correct OTP, it displays the message '**Ad is successfully posted**' else it displays the message '**Wrong Verification code! Ad is not posted**'

**Note:** Refer the figures corresponding to the steps.

**Log in**

[Login with phone & OTP](#)  
(or)  
[Log in using your Friendsbook account](#)

Figure 1

Login with phone & OTP

Phone number\*

**Get OTP**

Figure 2

Confirm Your Mobile Number

Enter the confirmation code that was texted to your phone

[Resend code](#)

**Confirm** CANCEL

Figure 3

 India's Largest Marketplace [My Account](#) [Submit a Free Ad](#)

Figure 4

**Submit a Free Classified Ad**

Ad Title*	Touch pad Samsung Mobile		
Category*	Mobiles » Mobile Phones » Samsung		
Price*	₹ 2000		
Ad Description*	Samsung - Model 16802 Year of purchase 2012 Touch Pad, touch Stick gorilla glass 13 px front camera 6 px rear camera 1 GB Internal memory		
Upload Photos		<input type="button" value="+"/>	<input type="button" value="+"/>
Name*	Anita Tiwari		
Phone number*	+919270896678		
Enter a city*	Pune, Maharashtra		
locality (nearby)	Swargate		
<b>Submit</b>			

Figure 5

#### Confirm Your Mobile Number

please enter the verification code sent to  
9270896678 to post your Ad

[Resend code](#)

CANCEL

Figure 6

### 3.3 Rules:

1. The phone number should accept only 10-digit number
2. The OTP should accept only 4-digit number
3. The fields marked as \* are mandatory fields on the form
4. Ad Title should accept maximum 70 characters and minimum 15 characters
5. The Category should accept only values given in following table

Category	Sub Category	Sub Category
Mobiles	Mobile phones	Iphone, Karbon, Lava, LG, Samsung, Micromax
	Tablets	Asus, Samsung, Appel I Pad
Cars	Cars	Toyota, Chevrolet, Zen, Swift
	Spare parts	Car glass, amplifier, Stereo system
	Beds	Single bed, double bed
	Wardrobes	Single door, two door, sliding door

6. Price accepts only digits in Rupees format
7. Ad Description should accept maximum 70 characters and minimum 15 characters
8. Upload Photos should allow browsing the pictures from your local machine. Minimum one photo should be uploaded.
9. Name should accept only characters
10. Select the City from the drop down box. This will automatically populate the Locality based on the selected City. Refer the below given table for sample valid values of Cities and Localities combination.

City	Locality
Pune, Maharashtra	Swargate
	Hinjewadi
	Shaniwarwada
Banglore, Karnataka	Whitefield
	Mathalli
	Bujigiri
Mumbai, Maharashtra	MG Road
	Dadar
	Nalasopara

## Lab 4.Creating Test Cases – Customer Complaint Form

<b>Goals</b>	<ul style="list-style-type: none"> <li>• Read “Customer Complaint Form Instructions” before starting the assignment.</li> <li>• Understand the application and develop creative test cases.</li> </ul>
<b>Time</b>	60 minutes

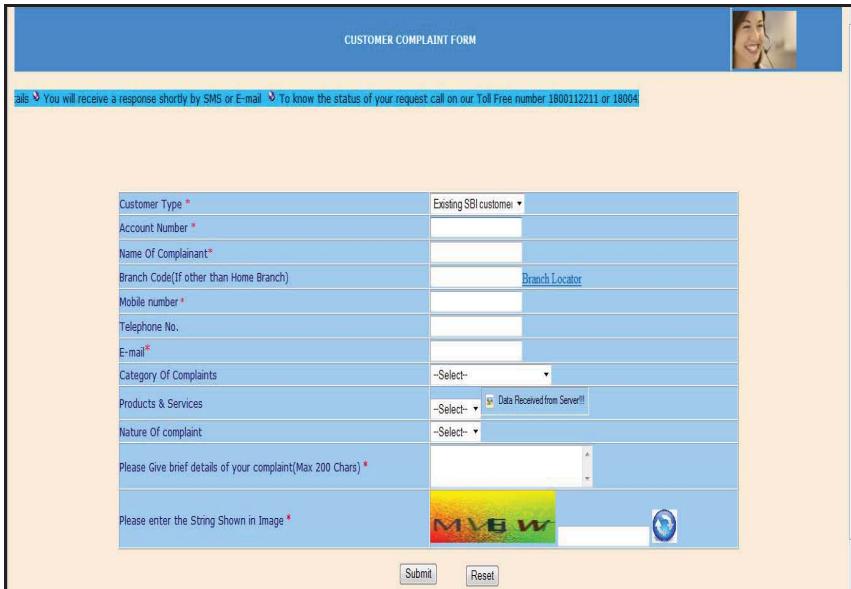
### 4.1 Case Study : Customer Complaint Form

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. At the top right is a small portrait of a woman. Below the title 'CUSTOMER COMPLAINT FORM' is a message: '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 the following fields:

- Customer Type \*: Existing SBI customer
- Account Number \*
- Name Of Complainant \*
- Branch Code[If other than Home Branch]: Branch Locator
- Mobile number \*
- Telephone No.
- E-mail \*
- Category Of Complaints: -Select-
- Products & Services: -Select- Data Received from Server!!
- Nature Of complaint: -Select-
- Please Give brief details of your complaint(Max 200 Chars) \*
- Please enter the String Shown in Image \*: CAPTCHA showing 'MVBW'

At the bottom 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.

## Lab 5.Creating Test Cases – Conference Room Booking

<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>	90 minutes

### 5.1: 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.igate.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:

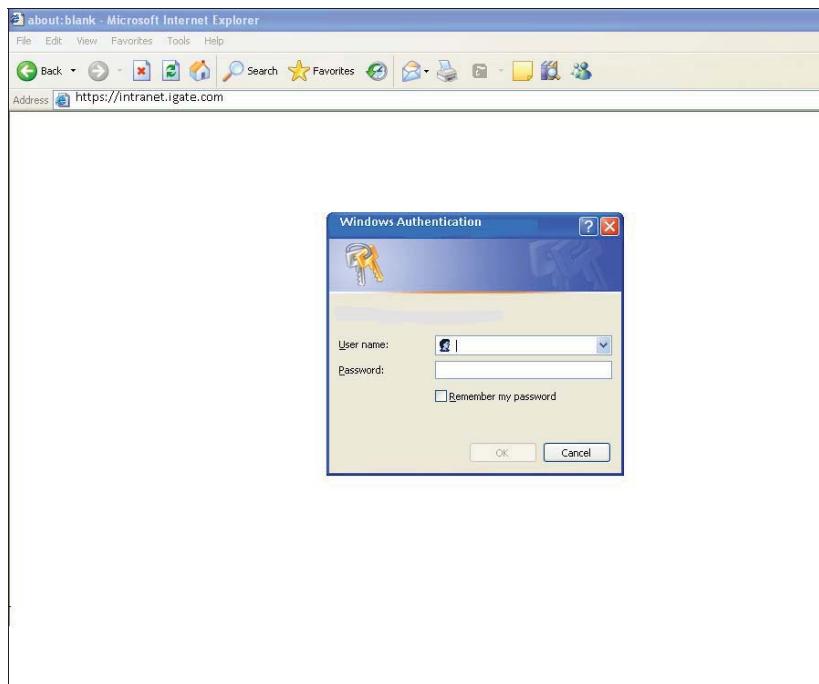
1. If you belong to a Non-GE Business Unit, you can book conference rooms only in Non-GE areas. However if you belong to a GE Business unit, you can book conference rooms at any location.
2. The employee Email id field is automatically filled with the email id of the person logged in.
3. Select the location 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.igate.com>



## Intranet home → Employee Corner → Conference Room Booking



Intranet

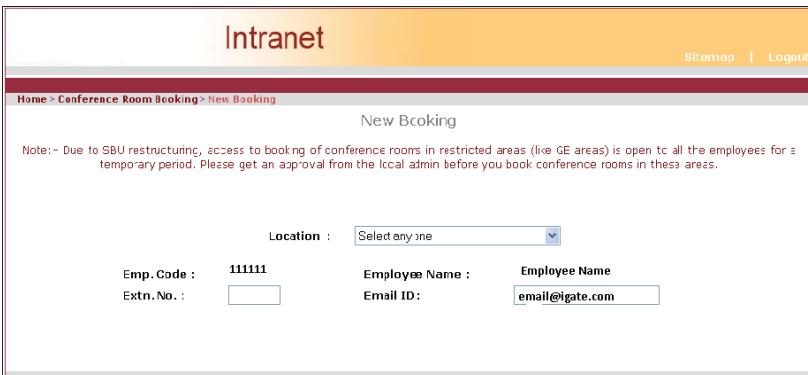
Sitemap | Logout

Home > Employee Self Service

**Menu**

- ◆ US Payroll - Electronic pay slip [Help](#)
- ◆ Conference Room Booking
- ◆ Global HelpDesk
- ◆ EPFO-NSS\ 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



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 :

Emp. Code : **111111** Employee Name : **Employee Name**  
Extn. No. :  Email ID: **email@igate.com**

## 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 :	SEEPZ	Sub Location :	SDFI-First Floor										
Date (mm/dd/yyyy) :	07 / 07 / 2005												
AVAILABLE ROOMS													
Time From	Time To	Description											
00:00:01	23:59:59	Main conference room											
AVAILABLE DEVICES													
Time From	Time To	Spider	Speaker	OHP	LCD/Video Projector								
00:00:01	23:59:59	2	1	2	1								
Emp. Code :	31608	Employee Name :	Jyoti Suresh										
Extn. No. :		Email ID :	jyotisuresh@igate.com										
Select Room :	Main conference room												
Description of the meeting :													
Time From (hh:mm):	01	:	00	AM	Time To (hh:mm):	01	:	00	AM				
Communication Device :						Visual Device :							
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 (mm/dd/yyyy)	Time From	Time To	Room Description	Name	Extn. No.	Communication Device	Visual Device
<input type="checkbox"/>	SEEPZ	SDPII - First Floor	7/7/2005	15:00:00	17:00:00	Main conference room	Jyoti Suresh	5204	speaker	LCD/Video Projector

**Delete**

## Lab 6: Creating Test Cases – Cyber Shopee

Goals	<ul style="list-style-type: none"><li>• Read the “CyberShopee” System documentation before starting the assignment.</li><li>• Develop creative test cases for the CyberShopee Online Shopping website.</li></ul>
Time	180 minutes

You need to go through the given case study carefully and write the appropriate test cases using different testing techniques learnt so far.

### 6.1 Case Study: CyberShoppe

“CyberShopee” is a web based application and can be accessed over the internet. Using this application a user can shop online for different products.

This online shopping website facilitates user to shop various products such as TV, Camera etc. belonging to different categories.

Following is the complete list of functionalities of the system. You can make appropriate assumptions wherever necessary and proceed.

There are four categories of users who would access the system viz. **Admin, Customer, Dealer and Delivery**. Each one of these users would have some exclusive privileges to be exercised on this website.

#### Note: Delivery user is out of scope.

Following are the functionalities to be performed by the **Admin** user.

1. Register on the website
2. Login
3. Approve Dealer
4. Approve Agency
5. Add Category
6. Add Sub Category

Following are the functionalities to be performed by the **Customer**.

1. Register on the website
2. Login
3. View Product Details
4. Purchase Products
5. Place the Order
6. view Account

## 7. Change Password

Following are the functionalities to be performed by the **Dealer**.

1. Register on the website
2. Login
3. Add Agency
4. Perform Dealer Operations

**Note:** Other than First Time registration, the other operations are allowed only for the registered users.

Once the Order is confirmed, Cyber Shopee will deliver the ordered products to the customer on his address specified during the registration. The Payment mode will be Cash on Delivery, Debit Card, Credit Card, NetBanking etc.

### 6.2 Process Work Flow:

#### 6.2.1 Visit CyberShopee Website

Visit <https://cyberShopeesystem.com> link through internet. This will take the user to the Home page of the website as given below. Refer to **Figure 1.1**.



Figure 1.1 CyberShopee Home Page

#### 6.2.2 Register

Purpose	To register with CyberShopee website
---------	--------------------------------------

**Functionality**

- User can register on the website.

As mentioned earlier, the user need to register on the website to avail different facilities provided. The user needs to fill in the registration form to do so. Refer to the **Figure 1.2**. Once the user has been successfully registered on the website, he can login to the website with the registered username & password and proceed.



The screenshot shows a registration form for 'CyberShoppee'. At the top right is a shopping cart icon containing several colored circles. To its left is the 'CyberShoppee' logo. Below the logo is a 'Logout' link. The form itself has fields for User Id, Full Name, Address, E-Mail, Password, Confirm Password, and Contact Number. There is also a 'Role' dropdown menu set to 'Admin' and a 'Submit' button. At the very bottom of the form area, it says 'Copyright. All rights reserved.'

**Figure 1.2 Register****Requirements:**

- All fields are mandatory.
- User Id should contain only 6 characters.
- Full Name should contain upto 30 characters and begin with uppercase letter.
- Address should contain 200 alphanumeric characters.
- Email should accept only valid email address e.g. someone@gmail.com
- Password should contain at least one uppercase character and one special character.
- Contact Number should contain only 10 digits and begin with 7/ 8/ 9.
- Role should be populated with these options (Admin, Customer, Dealer and Delivery).

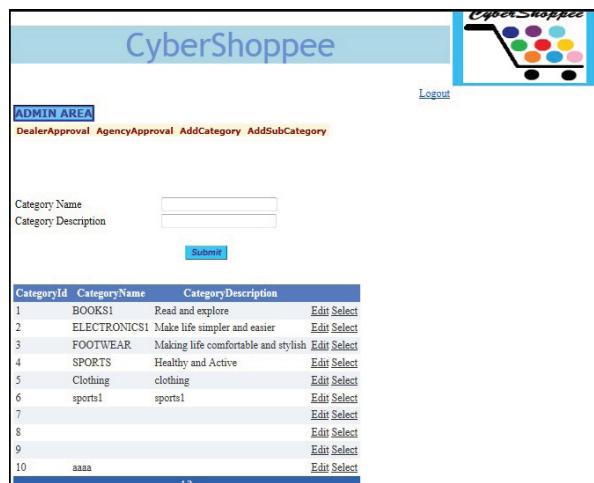
### 6.2.3 Admin Module

Note: Dealer approval and Agency Approval not in scope.



Figure 1.3 Admin Home Page

#### a. Add Category



CategoryId	CategoryName	CategoryDescription	
1	BOOKS1	Read and explore	Edit Select
2	ELECTRONICS1	Make life simpler and easier	Edit Select
3	FOOTWEAR	Making life comfortable and stylish	Edit Select
4	SPORTS	Healthy and Active	Edit Select
5	Clothing	clothing	Edit Select
6	sports1	sports1	Edit Select
7			Edit Select
8			Edit Select
9			Edit Select
10	aaaa		Edit Select

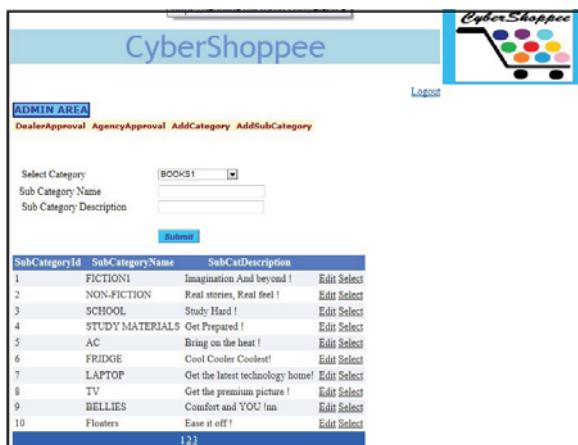
Figure 1.4 Add Category Page

Purpose	To add product categories
---------	---------------------------

<b>Functionality</b>	<ul style="list-style-type: none"> <li>Admin can add product category</li> <li>Admin can edit product category</li> <li>Admin can update product category</li> </ul>
----------------------	--

**Requirements:**

1. Category Name and Category Description cannot be blank.
2. Admin can edit any information of existing category & it should be successfully updated.

**b. Add Sub Category**


SubCategoryId	SubCategoryName	SubCatDescription	Edit	Select
1	FICTION1	Imagination And beyond !	<a href="#">Edit</a>	<a href="#">Select</a>
2	NON-FICTION	Real stories, Real feel !	<a href="#">Edit</a>	<a href="#">Select</a>
3	SCHOOL	Study Hard !	<a href="#">Edit</a>	<a href="#">Select</a>
4	STUDY MATERIALS	Get Prepared !	<a href="#">Edit</a>	<a href="#">Select</a>
5	AC	Bring on the heat !	<a href="#">Edit</a>	<a href="#">Select</a>
6	FRIDGE	Cool Cooler Coolerr!	<a href="#">Edit</a>	<a href="#">Select</a>
7	LAPTOP	Get the latest technology home!	<a href="#">Edit</a>	<a href="#">Select</a>
8	TV	Get the premium picture !	<a href="#">Edit</a>	<a href="#">Select</a>
9	BELLIES	Comfort and YOU 'ma	<a href="#">Edit</a>	<a href="#">Select</a>
10	Flooters	Ease it off !	<a href="#">Edit</a>	<a href="#">Select</a>
123				

**Figure 1.5 Add Sub Category Page**

<b>Purpose</b>	To add product sub category
<b>Functionality</b>	<ul style="list-style-type: none"> <li>Admin can add product sub category</li> <li>Admin can update product sub category</li> </ul>

**Requirements:**

1. The select category dropdown box should auto populate with the existing category ids.
2. User need to select category id
3. User need to fill in the subcategory name and subcategory description & both fields cannot be left blank.

4. Admin can edit any information of a product subcategory using same page.
5. User need to click on the “Edit” link to update the details of existing subcategory.
6. Once user has edited the details, he can confirm the same by clicking on the Update link.

#### 6.2.4 Customer Module

##### a. Search Products



The screenshot shows the CyberShoppee customer area search interface. At the top, there's a navigation bar with links for Search Products, My Orders, My Cart, Change Password, and My Account. Below the navigation is a 'CUSTOMER AREA' section. It includes dropdown menus for 'Select Category' (set to BOOKS) and 'Select Subcategory' (set to FICTION), and a 'Search' button. A table below lists products, with one row highlighted for 'Wings of Fire'. The table columns are: ProductId, SubCategoryId, ProductName, ModelNo, UnitCost, ProductType, TargetCustomer, ProductDescription, DealerId, and ImageUrl. The highlighted row shows ProductId 1, SubCategoryId 1, ProductName 'Wings of Fire', ModelNo 'FC0001', UnitCost 300, ProductType Fiction, TargetCustomer All, DealerId 'A.P.J.Abdul Kalam', and ImageUrl 'shb194'. At the bottom of the page, there's a copyright notice: 'Copyright. All rights reserved.'

Figure 1.6 Search Product Page



Figure 1.7 Add To Cart Page

Purpose	To search product
Functionality	<ul style="list-style-type: none"> <li>Customer can search for a product</li> <li>Customer can then add the searched product in the shopping cart</li> </ul>

#### Requirements:

1. Customer need to select the category.
2. Based on category selected the subcategory will be populated.
3. Customer need to click on the Search button to initiate the search of the product.
4. If the matching product exists, the details will be displayed in the table below.
5. Customer can click on View Product link to view the selected product and at the same time he can also add this product to the cart by clicking on the Add To Cart button.
6. When user clicks on "Add To Cart" button the product should get added in to the cart.
7. Select Quantity field cannot be null.

8. When user clicks on “Cancel” button user should navigate to Search Product page.

**b. My Orders**



OrderId	CustomerId	ShippingAddress	TotalCost	TotalItems	OrderDate
32	cust00	Pune	600	2	6/15/2016 12:00:00 AM

[View Details](#)   [Order Details](#)

Copyright. All rights reserved.

Figure 1.8 My Orders Page

<b>Purpose</b>	To view orders
<b>Functionality</b>	<ul style="list-style-type: none"> <li>Customers can view orders placed by him</li> </ul>

**Requirements:**

- Customer can click on My Orders tab to view his orders.
- If the cart is null, and user clicks on “Place Order” Button, the order should not get added into My Orders.

**c. My Cart**



RecordId	CartId	ProductId	Quantity	Cost	DateCreated
14	cust00	1	4	1200	6/15/2016 12:00:00 AM

[Place Order](#)   [Remove From Cart](#)

Copyright. All rights reserved.

Figure 1.9 My Cart Page

<b>Purpose</b>	To view products in cart and place order
<b>Functionality</b>	<ul style="list-style-type: none"> <li>• Customer can add products in cart</li> <li>• Customer can place order</li> </ul>

**Requirements:**

1. Customer can place order for selected items from the cart.
2. Customer can remove selected items from the cart.

**d. Change Password**


**Figure 1.10 Change Password Page**

<b>Purpose</b>	To change password
<b>Functionality</b>	<ul style="list-style-type: none"> <li>• Customer can change password</li> </ul>

**Requirements:**

1. Customer can click on "Change" button and change the password.
2. Customer can click on "Exit" button to come out from current page and navigate to customer's home page..
3. New password and Confirm password should match with each other.
4. If Old password is incorrect, system should alert customer with proper error message.

### e. My Account



**Figure 1.11 My Account Page**

<b>Purpose</b>	To Edit account Details
<b>Functionality</b>	<ul style="list-style-type: none"> <li>Customer can edit his account details.</li> </ul>

#### Requirements:

1. Customers are restricted to edit “Customer Id”.
2. Customer Name should contain only characters and begin with uppercase letter.
3. Email should be in `someone@gmail.com` format.
4. Contact Number should contain only 10 digits and begin with 7 or 8 or 9.
5. Customer can click on “Edit profile” to edit the profile.
6. Customer can click on “Save Changes” button to save edited profile.
7. Customer can click on “Cancel” button to cancel the changes and should navigate to Customer Home page.

#### 6.2.5 Dealer Module

##### a. Dealer Operations

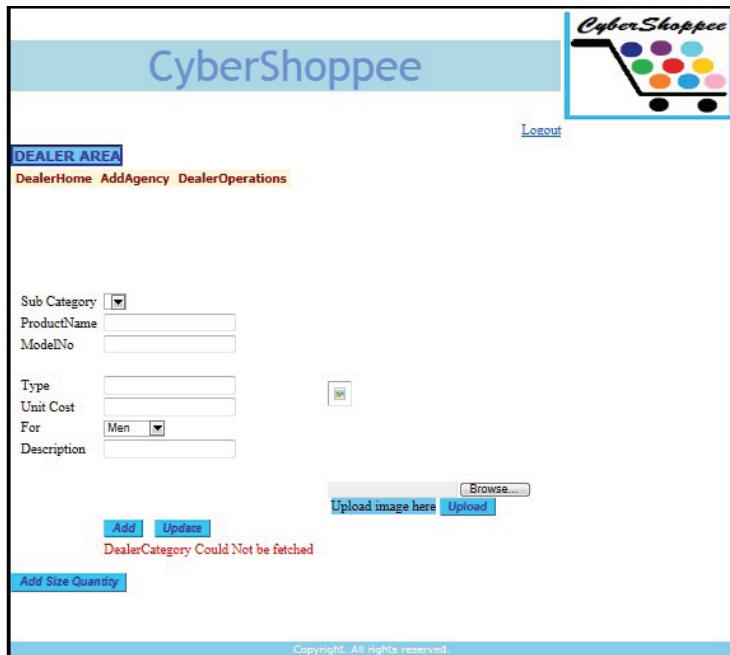


Figure 1.12 Add Product Details

Purpose	To add product details
Functionality	<ul style="list-style-type: none"> <li>• Dealer can add product details.</li> <li>• Dealer can update product details.</li> </ul>

**Requirements:**

1. All fields are mandatory.
2. Sub category dropdown box should auto populate.
3. Product Name can be alphanumeric.
4. Model Number can be alphanumeric.
5. Unit cost should contain only digits.
6. Description should contain 100 characters.
7. Dealer can click on browse button and able to upload the image.
8. Dealer can click on "Add" button to add the products.
9. Dealer can click on "Update" button to update the products.
10. Dealer can click on "Add Size Quantity" button to add the product quantity.