| 2017-18 Onwards (MR-17) | MALLA REDDY ENGINEERING COLLEGE (Autonomous) | B.Tech. VII Semester | | |
|---|---|---|---|---|
| Code:70617 | SOFTWARE TESTING METHODOLOGIES LAB | L | T | P |
| Credits: 2 | | - | - | 4 |

**Prerequisites:** Software Engineering

**Course Objectives:**
This Course enables the students to understand the principles and need for various types of testing test adequacy assessment using: data flow, transaction flow and path testing, describe strategies for generating system test cases, apply the essential characteristics of path product and regular expressions, and explain about the people, organizational issues in Testing.

**Software Requirements:** Turbo C
**List of Programs:**
1 Write programs in 'C' Language to demonstrate the working of the following constructs:

i) do...while

ii) while….do

iii) if…else

iv) switch

v) for

2. "A program written in 'C' language for Matrix Multiplication fails" Introspect the causes for its failure and write down the possible reasons for its failure.

3. Take any system (e.g. ATM system) and study its system specifications and report the various bugs.

4. Write the test cases for any known application (e.g. Banking application)

5. Create a test plan document for any application (e.g. Library Management System)

6. Study of Win Runner Testing Tool and its implementation

a) Win runner Testing Process and Win runner User Interface.

b) How Win Runner identifies GUI(Graphical User Interface) objects in an application and describes the two modes for organizing GUI map files.

c) How to record a test script and explains the basics of Test Script Language (TSL).

7. Implement Win runner and perform the following functionalities

a) How to synchronize a test when the application responds slowly.

b) How to create a test that checks GUI objects and compare the behaviour of GUI objects in different versions of the sample application.

8. Implement Win runner and perform the following functionalities

a) How to create and run a test that checks bitmaps in your application and run the test on different versions of the sample application and examine any differences, pixel by pixel.

b) How to Create Data-Driven Tests which supports to run a single test on several sets of data from a data table.

9. Implement the following using Win Runner

a) How to read and check text found in GUI objects and bitmaps.

b) How to create a batch test that automatically runs the tests.

10. Implement the following using Win Runner

How to update the GUI object descriptions which in turn supports test scripts as the application changes.

11. Apply Win Runner testing tool implementation in any real time applications.

12. Study of any test management tool and any open source testing tool

**TEXTBOOKS:**

1. Boris Beizer, "Software Testing Techniques",Dream tech Press, 2003.

**REFERENCES:**

1. Renu Rajni,"**Software Testing; Effective Methods Tools and Testing**", PHI.
2. Srinivasan Desikan, "**Software Testing Principles and Practices**"

**Course Outcomes:**

At the end of the course, students will be able to

1. **Solve** specific problems alone or in teams.
2. **Manage** a project from beginning to end.
3. **Understand** team management.
4. **Define**, formulate and analyze a problem.

# Malla Reddy Engineering College (A)

## Department of Information Technology

### III B. Tech I SEM (MR17)

## Software Testing Methodologies Lab Manual

**OBJECTIVES OF THE LAB**

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as yet undiscovered error.
3. A successful test is one that uncovers an as yet undiscovered error.

**REQUIREMENTS**

Server System configuration : 8 GB RAM , 500 MB of free disk space, Win 2K3 server, IIS 6.0, MSAccess/Oracle 7.x,8.x,9/MS SQL

ServerClient System configuration : 2 GB RAM , 10 MB of free disk space, Win XP, IE 6.0

## Testing Lab List of Experiments

1. Write programs in „C" Language to demonstrate the working of the following a. constructs: i) do...while ii) while….do iii) if…else iv) switch v) for
2. A program written in „C" language for Matrix Multiplication fails‖ Introspect the causes for its failure and write down the possible reasons for its failure.
3. Take any system (e.g. ATM system) and study its system specifications and report the various bugs.
4. Write the test cases for any known application (e.g. Banking application)
5. Create a test plan document for any application (e.g. Library Management System)
6. Study of any testing tool (e.g. Win runner)
7. Study of any web testing tool (e.g. Selenium)
8. Study of any bug tracking tool (e.g. Bugzilla, bugbit)
9. Study of any test management tool (e.g. Test Director)
10. Study of any open source-testing tool (e.g. Test Link)

**1. Write a „c‟ program to demonstrate the working of the fallowing constructs:**
**i)do…while ii) while…do iii) if …else iv) switch v) for Loops in C language**

**//A. AIM: To demonstrate the working of do..while construct**

**Objective**

To understand the working of do while with different range of values and test cases

```
#include <stdio.h>
void main (){
        int i, n=5,j=0; clrscr();
        printf(―enter a no‖);

        scanf(―%d‖,&i);

    do{
                if(i%2==0) {
                        printf("%d", i);
                        printf("is a even no.");
                        i++;
                        j++;
                    }
                else        {
                                printf("%d", i);
                                printf("is a odd no.\n");
                                i++;
                                j++;
                            }
                }while(i>0&&j<n);
                    getch();

}
```

| Input | Actual output |
|---|---|
| 2 | 2 is even number<br>3 is odd number<br>4 is even number<br>5 is odd number<br>6 is even number |

**Test cases:**

**Test case no: 1**
**Test case name**: Positive values within range

| Input =2 | Expected output | Actual output | Remarks |
|---|---|---|---|
| | 2 is even number | 2 is even number | |
| | 3 is odd number | 3 is odd number | success |
| | 4 is even number | 4 is even number | |
| | 5 is odd number | 5 is odd number | |
| | 6 is even number | 6 is even number | |

**Test case no:2**
**Test case name:** Negative values within a range

| **Input = -2** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| | -2 is even number | -2 is an even number | |
| | -3 is odd number | | fail |
| | -4 is even number | | |
| | -5 is odd number | | |
| | -6 is even number | | |

**Test case no: 3**

**Test case name:** Out of range values testing

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| 123456789122222222222 | 12345678912222222213 | 234567891222222215 | fail |

**//B. Aim:To demonstrate the working of while construct**

**Objective**

To understand the working of while with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>
void main (){
        int i, n=5,j=1; clrscr();
        printf(—enter a no||);
        scanf(—%d||,&i);
        while (i>0 && j<n){
                if(i%2==0){
                        printf(—%d||,i);
                        printf(—is a even
                                number||; i++;
                                        j++;
                                }
                else{
                        printf(—%d||,i);
                        printf(—is a odd
                        number||); i++;
                        j++;
                    }}
getch();
}
```

| **Input** | **Actual output** |
|---|---|
| 2 | 2 is even number |
| | 3 is odd number |
| | 4 is even number |
| | 5 is odd number |
| | 6 is even number |

**Test cases:**

**Test case no: 1**
**Test case name**: Positive values within range

| **Input =2** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| | 2 is even number | 2 is even number | |
| | 3 is odd number | 3 is odd number | success |
| | 4 is even number | 4 is even number | |
| | 5 is odd number | 5 is odd number | |
| | 6 is even number | 6 is even number | |

**Test case no:2**
**Test case name:** Negative values within a range

| **Input = -2** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| | -2 is even number | -2 is an even number | |
| | -3 is odd number | | fail |
| | -4 is even number | | |
| | -5 is odd number | | |
| | -6 is even number | | |

**Test case no: 3**

**Test case name:** Out of range values testing

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| 1234567891222222222222 | 123456789122222222213 | 234567891222222215 | fail |

**//C. Aim: To demonstrate the working of if else construct**

**Objective**

To understand the working of if else with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>

void main (){

        int i;
        clrscr();
        printf(—enter a number
        ||); scanf(—%d||,&i);

        if(i%2==0){
                        printf(—%d||,i);
                        printf(—is a even number||);
                    }
         else{
```

```
                printf(—%d||,i);
                printf(—is a odd number||);
        }
getch();

}
```

| Input | Actual output |
|---|---|
| 2 | 2 is even number |
| | 3 is odd number |
| | 4 is even number |
| | 5 is odd number |
| | 6 is even number |

**Test cases:**

**Test case no: 1**

**Test case name**: Positive values within range

| Input =2 | Expected output | Actual output | Remarks |
|---|---|---|---|
| | 2 is even number | 2 is even number | |
| | 3 is odd number | 3 is odd number | success |
| | 4 is even number | 4 is even number | |
| | 5 is odd number | 5 is odd number | |
| | 6 is even number | 6 is even number | |

**Test case no:2**

**Test case name:** Negative values within a range

| Input = -2 | Expected output | Actual output | Remarks |
|---|---|---|---|
| | -2 is even number | -2 is an even number | |
| | -3 is odd number | | fail |
| | -4 is even number | | |
| | -5 is odd number | | |
| | -6 is even number | | |

**Test case no: 3**

**Test case name:** Out of range values testing

| Input | Expected output | Actual output | Remarks |
|---|---|---|---|
| 123456789122222222222 | 12345678912222222213 | 234567891222222215 | fail |

**// D. To demonstrate the working of switch construct**

**Objective**

To understand the working of switch with different range of values and test cases

```
void main() {

        int a,b,c;
        clrscr();
        printf(—1.Add/n 2.Sub /n 3.Mul /n 4.Div /n Enter Your
        choice||); scanf(—%d|| , &i);
```

```
printf(—Enter a,b values||);
scanf(—%d%d||,&a,&b);
switch(i){
        case 1: c=a+b;
                printf(— The sum of a & b is: %d||
                ,c); break;
        case 2: c=a-b;
                printf(— The Diff of a & b is: %d||
                ,c); break;
        case 3: c=a*b;
                printf(— The Mul of a & b is: %d||
                ,c); break;
        case 4: c=a/b;
                printf(— The Div of a & b is: %d||
                ,c); break;
        default:
                printf(— Enter your
                choice||); break;
    }
getch();
}
```

**Output:**

| **Input** | **Output** |
|---|---|
| Enter Ur choice: 1<br>Enter a, b Values: 3, 2 | The sum of a & b is:5 |
| Enter Ur choice: 2<br>Enter a, b Values: 3, 2 | The diff of a & b is: 1 |
| Enter Ur choice: 3<br>Enter a, b Values: 3, 2 | The Mul of a & b is: 6 |
| Enter Ur choice: 4<br>Enter a, b Values: 3, 2 | The Div of a & b is: 1 |

**Test cases:**

**Test case no: 1**
**Test case name**: Positive values within range

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Enter Ur choice: 1<br>Enter a, b Values: 3, 2 | The sum of a & b is:5 | 5 | |
| Enter Ur choice: 2<br>Enter a, b Values: 3, 2 | The diff of a & b is: 1 | 1 | Success |

Enter Ur choice: 3
Enter a, b Values: 3, 2 The Mul of a & b is: 6 6

Enter Ur choice: 4
Enter a, b Values: 3, 2 The Div of a & b is: 1 1

**Test case no:2**

**Test case name:** Out of range values testing

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Option: 1<br>a= 22222222222222<br><br>b=22222222222222 | 44444444444444 | -2 | fail |

**Test case no: 3**

**Test case name:** Divide by zero

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Option: 4<br><br>a= 10 & b=0 | error | | fail |

**// E. Aim: To demonstrate working of for construct**

**Objective**

To understand the working of for with different range of values and test cases

```
#include <stdio.h>
#include <conio.h>

void main (){ int
        i;
        clrscr();
        printf(—enter a no||);
        scanf(—%d||,&i);
                for(i=1;i<=5;i++){
                        if(i%2==0){
                                printf(—%d||, i);
                                printf(— is a even
                                no||); i++;
                                    }
                        else{
                                printf(—%d||, i);
                                printf(— is a odd
```

```
                    no||); i++;
                }
        }
        getch();
}
```

**Output:**

Enter a no: 5

0 is a even no
1 is a odd no
2 is a even no
3 is a odd no
4 is a even no
5 is a odd no

**Test cases:**

**Test case no: 1**
**Test case name**: Positive values within range

| **Input =2** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| | 0 is even number | 0 is even number | |
| | 1 is odd number | 1 is odd number | success |
| | 2 is even number | 2 is even number | |

**Test case no:2**
**Test case name:** Negative values within a range

| **Input = -2** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| | 0 is even number | 0 is an even number | |
| | -1 is odd number | -1 is even no | fail |
| | -2 is even number | -2 is odd no | |

**Test case no: 3**

**Test case name:** Out of range values testing

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| 123456789122222222222 | 12345678912222222213 | 234567891222222215 | fail |

**2. Aim: A program written in c language for matrix multiplication fails "Introspect the causes for its failure and write down the possible reasons for its failure".**

**Objective: Understand the failures of matrix multiplication**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3],b[3][3],c[3][3],i,j,k,m,n,p,q;
clrscr();
printf(— Enter 1st matrix no.of rows &
cols||) scanf(—%d%d||,&m,&n);
printf(— Enter 2nd matrix no.of rows &
cols||) scanf(—%d%d||,&p,&q);

printf("\n enter the matrix elements");
for(i=0;i<m;i++);
{
for(j=0;j<n;j++);
{
scanf("%d",&a[i][j]);
}
}

printf("\n a matrix
is\n"); for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
scanf("%d\t",&b[i][j]);
}
}
printf("\n b matrix is\n");

for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
```

```
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
c[i][j]=0;
for(k=0;k<n;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
}
}

for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",c[i][j]);

}
printf("\n");
}

getch();
}
```

Output:

```
Enter Matrix1: 1 1 1
               1 1 1
               1 1 1

Enter Matrix2: 1 1 1
               1 1 1
               1 1 1


Actual Output : 3 3 3
                3 3 3
                3 3 3
```

**Test cases:**

**Test case no: 1**
**Test case name**: Equal no.of rows & cols

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Matrix1 rows & cols= 3 3<br>Matrix2 rows & cols= 3 3 | | | |

| | | | | |
|---|---|---|---|---|
| Matrix1: | 1 1 1 | | | |
| | 1 1 1 | 3 3 3 | 3 3 3 | |
| | 1 1 1 | 3 3 3 | 3 3 3 | Success |
| | | 3 3 3 | 3 3 3 | |
| Matrix2: | 1 1 1 | | | |
| | 1 1 1 | | | |
| | 1 1 1 | | | |

**Test case no:2**

**Test case name:** Cols of 1st matrix not equal to rows of 2nd matrix

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Matrix1 rows & cols= 2 2<br>Matrix2 rows & cols= 3 2 | Operation Can‘t be Performed | | fail |

**Test case no: 3**

**Test case name:** Out of range values testing

| **Input** | **Expected output** | **Actual output** | **Remarks** |
|---|---|---|---|
| Matrix1 rows & cols= 2 2<br>Matrix2 rows & cols= 2 2 | | | |
| 1234567891 2222222222<br>2234567891 2222222221 | | | fail |
| 234567891 22222221533<br>213242424 56456475457 | | | |

## 3. Aim: Take any system (e.g. ATM system) and study its system specifications and report the various bugs.

**Program:**

**Features to be tested:**

1. Validity of the card.
2. Withdraw Transaction flow of ATM.
3. Authentication of the user's.
4. Dispense the cash from the account.
5. Verify the balance enquiry.
6. Change of PIN number.

**Bugs Identified:**

| Bug-Id | Bug Name |
|---|---|
| ATM_001 | Invalid Card |
| ATM_002 | Invalid PIN |
| ATM_003 | Invalid Account type |
| ATM_004 | Insufficient Balance |
| ATM_005 | Transaction Limit |
| ATM_006 | Day limit |
| ATM_007 | Invalid money denominations |
| ATM_008 | Receipt not printed |
| ATM_009 | PIN change mismatch |

**Bug Report:**

**Bug Id:** ATM_001
**Bug Description:** Invalid card
**Steps to reproduce:** 1. Keep valid card in the ATM.
**Expected Result:** Welcome Screen
**Actual Result:** Invalid card
**Status :** Pass/Fail

**Bug Id:** ATM_002
**Bug Description:** Invalid PIN entered
**Steps to reproduce:**

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Menu screen should be displayed.
**Expected Result:** Menu screen displayed
**Actual Result:** Invalid PIN screen is displayed
**Status :** Pass/Fail

**Bug Id:** ATM_003
**Bug Description:** Invalid Account type selected.
**Steps to reproduce:**
1. Enter a valid user PIN number.
2. Select the withdraw option on the main menu.
3. Choose the correct type of account (either savings or current account).
**Expected Result:** Enter the Amount screen displayed
**Actual Result:** Invalid Account type screen is displayed.
**Status :** Pass/Fail

**Bug Id:** ATM_004
**Bug Description:** Insufficient Balance
**Steps to reproduce:**
1. Menu screen should be displayed.
2. Select the withdraw option.
3. Select the correct type of account.
4. Enter the sufficient amount to withdraw from the account.
5. Dispense the cash screen & amount to be deducted from account
**Expected Result:** Collect the amount screen displayed
**Actual Result:** Insufficient balance in the account
**Status :** Pass/Fail

**Bug Id:** ATM_005
**Bug Description:** Withdraw Limit per transaction.
**Steps to reproduce:**
1. Menu screen should be displayed.
2. Select the withdraw option.
3. Select the correct type of account.
4. Enter sufficient amount to withdraw from the account Transaction within the limit.
5. Dispense the cash screen & amount to be deducted from account.
**Expected Result:** Cash is dispensed and collect the receipt
**Actual Result:** Transaction limit exceeded screen is displayed
**Status :** Pass/Fail

**Bug Id:** ATM_006
**Bug Description:** Withdraw limit per day
**Steps to reproduce:**
1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Enter the amount to withdraw from the account.

4. Amount enter is over the day limit (>40000)
5. Amount enter is over the day limit and display screen is displayed.
**Expected Result:** Cash is dispensed and collect the receipt.
**Actual Result:** Day limit exceeded screen is displayed.
**Status :** Pass/Fail

**Bug Id:** ATM_007
**Bug Description:** Amount enter denominations
**Steps to reproduce:**
1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Enter the amount which should be in multiples of 100.
4. Cash Dispenser screen is displayed.
**Expected Result:** Collect the amount screen is displayed.
**Actual Result:** Amount enter not in required denominations.
**Status :** Pass/Fail

**Bug Id:** ATM_008
**Bug Description:** Statement not printed
**Steps to reproduce:**
1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Select the mini statement.
4. Current balance is displayed on the screen.
5. Collect printed receipt of the statement.
**Expected Result:** Collect the mini statement receipt
**Actual Result:** receipt not printed.
**Status :** Pass/Fail

**Bug Id:** ATM_009
**Bug Description:** PIN mismatch
**Steps to reproduce:**
1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Select the change PIN option on the menu.
4. Enter the current PIN.
5. Enter the new PIN.
6. Retype the new PIN
7. PIN successfully changed displayed on the screen.
**Expected Result:** PIN change successful.
**Actual Result:** PIN mismatched due to wrong PIN entered
**Status :** Pass/Fail

| APPLICATION NAME | Test Case Id | Test Scenario | Test Case | Expected Result | Actual Result | StatuS | Test Data |
|---|---|---|---|---|---|---|---|
| SBI ONLINE BANKING | 1 | Validate the loginpage enter invalid/ wrong user name and valid password | Enter invalid user name and valid password in SBI online Banking login page | System should not allow the customer to login the SBI online Banking login page and it should displaythe message like ”please enter valid username and password” | Customer is not ableto login SBI online banking account | Pass | Ex:<br>UID:abcdef<br>PWD:xyz123 |
| | 2 | validate the login page enter invalid user name and invalid password | Enter invalid user name and invalid pass word in SBI online Banking loginpage | System should not allow the customer to login the SBI online Banking login page and it should displaythe message like " please enter valid username and password | Customer is not ableto login SBI online Banking account | Pass | Ex:<br>UID:abcd<br>PWD:xyz12 |
| APPLICATION | 3 | Validate the loginpage enter valid user name and invalid password | Enter valid user name and invalid password in SBI online Banking loginpage | System should allow the user to login the SBIonline Banking login page | Customer is logged in to SBI online Banking login page | Pass | Ex:<br>UID:abcdefg<br>PWD:xyz1234 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 4 | Validate the loginpage enter valid user name and valid password | Enter valid user name and valid password in SBI online Banking loginpage | System should allow the user to login the SBIonline Banking login page | Customer is logged into SBI online Banking login page | Pass | Ex:<br>UID:abcdefg<br>PWD:xyz<br>123 |
| | 5 | Validate the user information or detail in the | a) User should able to loginSBI login page<br>b) User should | a) User/customer should able to login | Customer is not ableto see phone or | fail | |
| | | profile page | able to clickon profile link<br>c) On clicking profile link uses should able to see all user details like<br>1) User /customer name<br>2) User/customer address details<br>3) User/customer phone number | SBI login page with valid<br>b) Customer shouldbe able to click profile link.<br>c) Customer shouldsee all the customer information once he clickingon profile hyperlink | mobile number | | |

## 5. Create a test plan document for any application (e.g. Library Management System)

### VERSION HISTORY

*[Provide information on how the development and distribution of the **Test Plan**, up to the final point of approval, was controlled and tracked. Use the table below to provide the version number, the author implementing the version, the date of the version, the name of the person approving the version, the date that particular version was approved, and a brief description of the reason for creating the revised version.]*

| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
|---|---|---|---|---|---|
| 1.0 | *<Author name>* | *<mm/dd/yy>* | *<name>* | *<mm/dd/yy>* | Test Plan draft |
| | | | | | |
| | | | | | |

### 1 INTRODUCTION

#### PURPOSE OF THE TEST PLAN DOCUMENT

*[Provide the purpose of the Test Plan Document. This document should be tailored to fit a particular project's needs.]*

The Test Plan document documents and tracks the necessary information required to effectively define the approach to be used in the testing of the project's product. The Test Plan document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and testing team. Some portions of this document may on occasion be shared with the client/user and other stakeholder whose input/approval into the testing process is needed.

### 2 COMPATIBILITY TESTING

#### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

#### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 3 CONFORMANCE TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 4 FUNCTIONAL TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |

| | | | |
|---|---|---|---|
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 5 PERFORMANCE TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 6 REGRESSION TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 7 SYSTEM TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 8 UNIT TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## 9 USER ACCEPTANCE TESTING

### TEST RISKS / ISSUES

*[Describe the risks associated with product testing or provide a reference to a document location where it is stored. Also outline appropriate mitigation strategies and contingency plans.]*

### ITEMS TO BE TESTED / NOT TESTED

*[Describe the items/features/functions to be tested that are within the scope of this test plan. Include a description of how they will be tested, when, by whom, and to what quality standards. Also include a description of those items agreed not to be tested.]*

| Item to Test | Test Description | Test Date | Responsibility |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

### TEST APPROACH(S)

*[Describe the overall testing approach to be used to test the project's product. Provide an outline of any planned tests.]*

### TEST REGULATORY / MANDATE CRITERIA

*[Describe any regulations or mandates that the system must be tested against.]*

### TEST PASS / FAIL CRITERIA

*[Describe the criteria used to determine if a test item has passed or failed its test.]*

### TEST ENTRY / EXIT CRITERIA

*[Describe the entry and exit criteria used to start testing and determine when to stop testing.]*

### TEST DELIVERABLES

*[Describe the deliverables that will result from the testing process (documents, reports, charts, etc.).]*

### TEST SUSPENSION / RESUMPTION CRITERIA

*[Describe the suspension criteria that may be used to suspend all or portions of testing. Also describe the resumption criteria that may be used to resume testing.]*

### TEST ENVIRONMENTAL / STAFFING / TRAINING NEEDS

*[Describe any specific requirements needed for the testing to be performed (hardware/software, staffing, skills training, etc).)]*

## TEST PLAN APPROVAL

The undersigned acknowledge they have reviewed the *<Project Name>* **Test Plan** document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

*[List the individuals whose signatures are required. Examples of such individuals are Business Steward, Technical Steward, and Project Manager. Add additional signature lines as necessary.]*

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

**Appendix A: References**

*[Insert the name, version number, description, and physical location of any documents referenced in this document. Add rows to the table as necessary.]*

The following table summarizes the documents referenced in this document.

| Document Name and Version | Description | Location |
|---|---|---|
| *<Document Name and Version Number>* | *[Provide description of the document]* | *<URL or Network path where document is located>* |

## Experiment 6

## Aim: Study of Any Testing Tool( WinRunner)

WinRunner is a program that is responsible for the automated testing of software. WinRunner is a Mercury Interactive's enterprise functional testing tool for Microsoft windows applications.

**Importance of Automated Testing:**

1. Reduced testing time
2. Consistent test procedures – ensure process repeatability and resource independence. Eliminates errors of manual testing
3. Reduces QA cost – Upfront cost of automated testing is easily recovered over the lifetime of the product
4. Improved testing productivity – test suites can be run earlier and more often
5. Proof of adequate testing
6. For doing Tedious work – test team members can focus on quality areas.

**WinRunner Uses:**

1. With WinRunner sophisticated automated tests can be created and run on an application.
2. A series of wizards will be provided to the user, and these wizards can create tests in an automated manner.
3. Another impressive aspect of WinRunner is the ability to record various interactions, and transform them into scripts. WinRunner is designed for testing graphical user interfaces.
4. When the user make an interaction with the GUI, this interaction can be recorded. Recording the interactions allows to determine various bugs that need to be fixed.

5. When the test is completed, WinRunner will provide with detailed information regarding the results. It will show the errors that were found, and it will also give important information about them. The good news about these tests is that they can be reused many times.
6. WinRunner will test the computer program in a way that is very similar to normal user interactions. This is important, because it ensures a high level of accuracy and realism. Even if an engineer is not physically present, the Recover manager will troubleshoot any problems that may occur, and this will allow the tests to be completed without errors.
7. The Recover Manager is a powerful tool that can assist users with various scenarios. This is important, especially when important data needs to be recovered.

The goal of WinRunner is to make sure business processes are properly carried out. WinRunner uses TSL, or Test Script Language.

**WinRunner Testing Modes**

***Context Sensitive***

Context Sensitive mode records your actions on the application being tested in terms of the GUI objects you select (such as windows, lists, and buttons), while ignoring the physical location of the object on the screen. Every time you perform an operation on the application being tested, a TSL statement describing the object selected and the action performed is generated in the test script. As you record, WinRunner writes a unique description of each selected object to a GUI map.

The GUI map consists of files maintained separately from your test scripts. If the user interface of your application changes, you have to update only the GUI map, instead of hundreds of tests. This allows you to easily reuse your Context Sensitive test scripts on future versions of your application.

To run a test, you simply play back the test script. WinRunner emulates a user by moving the mouse pointer over your application, selecting objects, and entering keyboard input. WinRunner reads the object descriptions in the GUI map and then searches in the application being tested for objects matching these descriptions. It can locate objects in a window even if their placement has changed.

*Analog*

Analog mode records mouse clicks, keyboard input, and the exact x- and y-coordinates traveled by the mouse. When the test is run, WinRunner retraces the mouse tracks. Use Analog mode when exact mouse coordinates are important to your test, such as when testing a drawing application.

**The WinRunner Testing Process**

Testing with *WinRunner* involves six main stages:

**1. Create the GUI Map**

The first stage is to create the GUI map so WinRunner can recognize the GUI objects in the application being tested. Use the RapidTest Script wizard to review the user interface of

your application and systematically add descriptions of every GUI object to the GUI map. Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.

**2. Create Tests**

Next is creation of test scripts by recording, programming, or a combination of both. While recording tests, insert checkpoints where we want to check the response of the application being tested. We can insert checkpoints that check GUI objects, bitmaps, and databases. During this process, WinRunner captures data and saves it as *expected results*—the expected response of the application being tested.

**3. Debug Tests**

Run tests in Debug mode to make sure they run smoothly. One can set breakpoints, monitor variables, and control how tests are run to identify and isolate defects. Test results are saved in the debug folder, which can be discarded once debugging is finished. When WinRunner runs a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in **If**, **While**, **Switch**, and **For** statements. We can use the **Syntax Check** options (**Tools >Syntax Check**) to check for these types of syntax errors before running your test.

**4. Run Tests**

Tests can be run in Verify mode to test the application. Each time WinRunner encounters a checkpoint in the test script, it compares the current data of the application being tested to the expected data captured earlier. If any mismatches are found, WinRunner captures them as *actual results*.

**5. View Results**

Following each test run, WinRunner displays the results in a report. The report details all the major events that occurred during the run, such as checkpoints, error messages, system messages, or user messages. If mismatches are detected at checkpoints during the test run, we can view the expected results nd the actual results from the Test Results window. In cases of bitmap mismatches, one can also view a bitmap that displays only the difference between the expected and actual results.

We can view results in the standard WinRunner report view or in the Unified report view. The WinRunner report view displays the test results in a Windows-style viewer. The Unified report view displays the results in an HTML-style viewer (identical to the style used for QuickTest Professional test results).

**6. Report Defects**

If a test run fails due to a defect in the application being tested, one can report information about the defect directly from the Test Results window.

This information is sent via e-mail to the quality assurance manager, who tracks the defect until it is fixed.

**Using Winrunner Window**

Before you begin creating tests, you should familiarize yourself with the WinRunner main window.

**To start WinRunner:**

Choose **Programs** > **WinRunner** > **WinRunner** on the **Start** menu.

The first time you start WinRunner, the Welcome to WinRunner window and the ―What's New in WinRunner‖ help open. From the Welcome window you can create a new test, open an existing test, or view an overview of WinRunner in your default browser.

CASE TOOLS & SOFTWARE TESTING LAB MANUAL

If you do not want this window to appear the next time you start WinRunner, clear the **Show on Startup** check box. To show the **Welcome to WinRunner** window upon startup from within WinRunner, choose **Settings > General Options**, click the **Environment** tab, and select the **Show Welcome screen** check box.

**The Main WinRunner Window**

The main WinRunner window contains the following key elements:

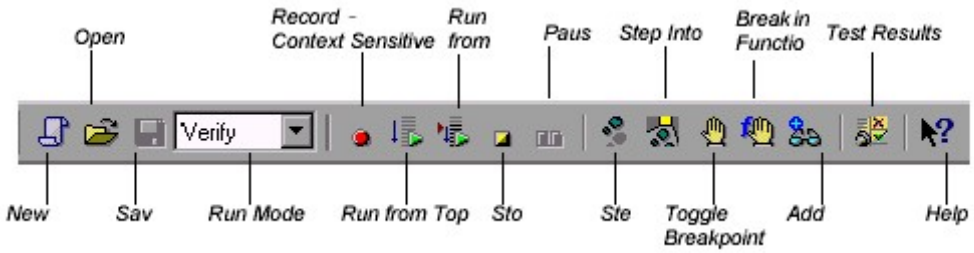
*WinRunner title bar*

*Menu bar,* with drop-down menus of WinRunner commands

*Standard toolbar,* with buttons of commands commonly used when running a test *User toolbar,* with commands commonly used while creating a test

*Status bar*, with information on the current command, the line number of the insertion point and the name of the current results folder

The *Standard toolbar* provides easy access to frequently performed tasks, such as opening, executing, and saving tests, and viewing test results.




**Standard Toolbar**

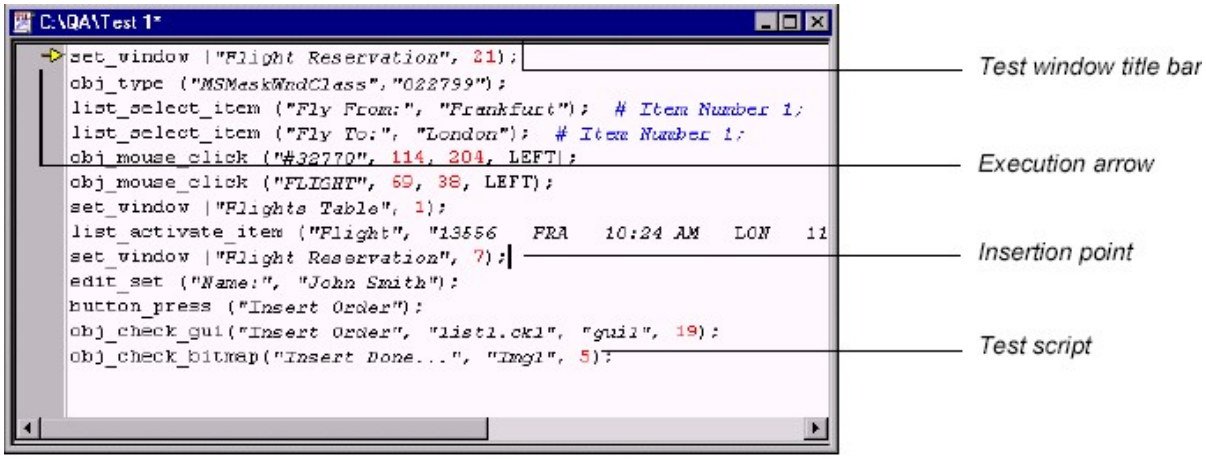
The *User toolbar* displays the tools you frequently use to create test scripts. By default, the User toolbar is hidden. To display the User toolbar, choose **Window** > **User Toolbar**. When you create tests, you can minimize the WinRunner window and work exclusively from the toolbar. The User toolbar is customizable. You choose to add or remove buttons using the **Settings > Customize User Toolbar** menu option. When you re-open WinRunner, the User toolbar appears as it was when you last closed it. The commands on the Standard toolbar and the User toolbar are described in detail in subsequent lessons.

Note that you can also execute many commands using *softkeys*. Softkeys are keyboard shortcuts for carrying out menu commands. You can configure the softkey combinations for your keyboard using the Softkey Configuration utility in your WinRunner program group. For more information, see the ―WinRunner at a Glance‖ chapter in your *WinRunner User's Guide*.

Now that you are familiar with the main WinRunner window, take a few minutes to explore these window components before proceeding to the next lesson.

**The Test Window**
You create and run WinRunner tests in the test window. It contains the following key elements:

- *Test window title bar*, with the name of the open test
- *Test script*, with statements generated by recording and/or programming in TSL, Mercury Interactive's Test Script Language
- *Execution arrow,* which indicates the line of the test script being executed during a test run, or the line that will next run if you select the Run from arrow option
- *Insertion point,* which indicates where you can insert or edit text




## Experiment 7:
## Study of any web testing tool (e.g. Selenium)

Selenium is a robust set of tools that supports rapid development of test automation for web-based applications. Selenium provides a rich set of testing functions specifically geared to the needs of testing of a web application. These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior.

- One of Selenium's key features is the support for executing one's tests on multiple browser platforms.
- **Selenium Components**
- Selenium is composed of three major tools. Each one has a specific role in aiding the development of web application test automation.

  Selenium-RC provides an API (Application Programming Interface) and library for each of its supported languages: HTML, Java, C#, Perl, PHP, Python, and Ruby. This ability to use Selenium-RC with a high-level programming language to develop test cases also allows the automated testing to be integrated with a project's automated build environment.

  **Selenium-Grid**

Selenium-Grid allows the Selenium-RC solution to scale for large test suites or test suites that must be run in multiple environments. With Selenium-Grid, multiple instances of Selenium-RC are running on various operating system and browser configurations; Each

of these when launching register with a hub. When tests are sent to the hub they are then redirected to an available Selenium-RC, which will launch the browser and run the test. This allows for running tests in parallel, with the entire test suite theoretically taking only as long to run as the longest individual test.

* Tests developed on Firefox via Selenium-IDE can be executed on any other supported browser via a simple Selenium-RC command line.

** Selenium-RC server can start any executable, but depending on browser security settings there may be technical limitations that would limit certain features.

**Flexibility and Extensibility**

Selenium is highly flexible. There are multiple ways in which one can add functionality to Selenium‘s framework to customize test automation for one‘s specific testing needs. This is, perhaps, Selenium‘s strongest characteristic when compared with proprietary test automation tools and other open source solutions. Selenium-RC support for multiple programming and scripting languages allows the test writer to build any logic they need into their automated testing and to use a preferred programming or scripting language of one‘s choice.

Selenium-IDE allows for the addition of user-defined ―user-extensions‖ for creating additional commands customized to the user‘s needs. Also, it is possible to re-configure how the Selenium-IDE generates its Selenium-RC code. This allows users to customize the generated code to fit in with their own test frameworks. Finally, Selenium is an Open Source project where code can be modified and enhancements can be submitted for contribution.

- .**Test Suites**
-
- A test suite is a collection of tests. Often one will run all the tests in a test suite as one continuous batch-job.
- When using Selenium-IDE, test suites also can be defined using a simple HTML file. The syntax again is simple. An HTML table defines a list of tests where each row defines the filesystem path to each test. An example tells it all.

<html>
- <head>
- <title>Test Suite Function Tests – Priority 1</title>
- </head> <body>
- <table>
- <tr><td>Suite Of Tests</td></tr>
- <tr><td><a href="./Login.html">Login</a></td></tr>
- <tr><td><a href="./SearchValues.html">Test Searching for Values</a></td></tr> <tr><td><a href="./SaveValues.html">Test Save</a></td></tr>

</table>
</body> </html>

A file similar to this would allow running the tests all at once, one after another, from the Selenium-IDE.

Test suites can also be maintained when using Selenium-RC. This is done via programming and can be done a number of ways. Commonly Junit is used to maintain a test suite if one is using Selenium-RC with Java. Additionally, if C# is the chosen language, Nunit could be employed. If using an interpreted language like Python with Selenium-RC than some simple programming would be involved in setting up a test suite. Since the whole reason for using Sel-RC is to make use of programming logic for

your testing this usually isn‘t a problem.
Few typical Selenium commands.

**open –** opens a page using a URL.

**click/clickAndWait –** performs a click operation, and optionally waits for a new page to load.

**verifyTitle/assertTitle –** verifies an expected page title.

**verifyTextPresent –** verifies expected text is somewhere on the page.

**verifyElementPresent –** verifies an expected UI element, as defined by its HTML tag, is present on the page.

**verifyText –** verifies expected text and it‘s corresponding HTML tag are present on the page.

**verifyTable –** verifies a table‘s expected contents.

**waitForPageToLoad –** pauses execution until an expected new page loads. Called automatically when clickAndWait is used.

**waitForElementPresent –** pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.

**Experiment 8**

**Aim: Study of Any Bug Tracking Tool (Bugzilla)**

Bugzilla is a ―Bug Tracking System‖ that can efficiently keep track of outstanding bugs in a product. Multiple users can access this database and query, add and manage these bugs. Bugzilla essentially comes to the rescue of a group of people working together on a product as it enables them to view current bugs and make contributions to resolve issues. Its basic repository nature works out better than the mailing list concept and an organized database is always easier to work with.

**Advantage of Using Bugzilla:**

1. Bugzilla is very adaptable to various situations. Known uses currently include IT support queues, Systems Administration deployment management, chip design and development problem tracking (both pre-and-post fabrication), and software and hardware bug tracking for luminaries such as Redhat, NASA, Linux-Mandrake, and VA Systems. Combined with systems such as CVS, Bugzilla provides a powerful, easy-to-use solution to configuration management and replication problems.

2. Bugzilla can dramatically increase the productivity and accountability of individual employees by providing a documented workflow and positive feedback for good performance. Ultimately, Bugzilla puts the power in user‘s hands to improve value to business while providing a usable framework for natural attention to detail and knowledge store to flourish.

The bugzilla utility basically allows to do the following:

Add a bug into the database
Review existing bug reports
Manage the content

Bugzilla is organised in the form of bug reports that give all the information needed about a particular bug. A bug report would consist of the following fields.

Product–>Component
Assigned to
Status (New, Assigned, Fixed etc)
Summary
Bug priority
Bug severity (blocker, trivial etc)
Bug reporter

**Using Bugzilla:**

Bugzilla usage involves the following activities

Setting Parameters and Default Preferences

- Creating a New User
- Impersonating a User
- Adding Products
- Adding Product Components
- Modifying Default Field Values
- Creating a New Bug
- Viewing Bug Reports

**Setting Parameters and Default Preferences:**

When we start using Bugzilla, we'll need to set a small number of parameters and preferences. At a minimum, we should change the following items, to suit our particular need:

▪ Set the *maintainer*

▪ Set the *mail_delivery_method*

▪ Set *bug change policies*

▪ Set the display order of bug reports

To set parameters and default preferences:

1. Click *Parameters* at the bottom of the page.
2. Under *Required Settings*, add an email address in the *maintainer* field.
3. Click *Save Changes*.
4. In the left side *Index* list, click *Email*.
5. Select from the list of mail transports to match the transport we're using. If evaluating a click2try application, select *Test*. If using SMTP, set any of the other SMTP options for your environment. Click *Save Changes*.
6. In the left side *Index* list, click *Bug Change Policies*.
7. Select On for *commentoncreate*, which will force anyone who enters a new bug to enter a comment, to describe the bug. Click *Save Changes*.
8. Click *Default Preferences* at the bottom of the page.
9. Select the display order from the drop-down list next to the *When viewing a bug, show comments in this order* field. Click *Submit Changes*.

**Creating a New User**

Before entering bugs, make sure we add some new users. We can enter users very easily, with a minimum of information. Bugzilla uses the email address as the user ID, because users are frequently notified when a bug is entered, either because they entered the bug, because the bug is assigned to them, or because they‘ve chosen to track bugs in a certain project.

To create a new user:

1. Click **Users**.
2. Click **add** a new user.
3. Enter the **Login name**, in the form of an email address.
4. Enter the **Real name**, a password, and then click **Add**.
5. Select the **Group access options**. we‘ll probably want to enable the following options in the row titled User is a member of these groups:

   *canconfirm*
   - *editbugs*
   - *editcomponents*

6. Click **Update** when done with setting options.

**Impersonating a User**

**Impersonating** a user is possible, though rare, that we may need to file or manage a bug in an area that is the responsibility of another user when that user is not available. Perhaps the user is on vacation, or is temporarily assigned to another project. We can impersonate the user to create or manage bugs that belong to that user.

**Adding Products**

We‘ll add a product in Bugzilla for every product we are developing. To start with, when we first login to Bugzilla, we‘ll find a test product called **TestProduct**. We should delete this and create a new product.

To add a product:

1. At the bottom of the page, click **Products**.
2. In the **TestProduct** listing, click **Delete**.
3. Click **Yes, Delete**.
4. Now click **Add a product**.
5. Enter a product name, such as ―Widget Design Kit.‖
6. Enter a description.

7. Click **Add**. A message appears that you‘ll need to add at least one component.

**Adding Product Components**

Products are comprised of components. Software products, in particular, are typically made up of many functional components, which in turn are made up of program elements, like classes and functions. It‘s not unusual in a software development team environment for different

individuals to be responsible for the bugs that are reported against a given component. Even if there are other programmers working on that component, it‘s not uncommon for one person, either a project lead or manager, to be the gatekeeper for bugs. Often, they will review the bugs as they are reported, in order to redirect them to the appropriate developer or even another team, to review the priority and severity supplied by the reporter, and sometimes to reject bugs as duplicates or enhancement requests, for example.

To add a component:

1. Click the link **add at least one component** in the message that appears after creating a new product.
2. Enter the **Component** name.
3. Enter a **Description**.
4. Enter a **default assignee**. Use one of the users we‘ve created. Remember to enter the assignee in the form of an email address.
5. Click **Add**.
6. To add more components, click the name of product in the message that reads edit other components of product <**product name**>.

**Modifying Default Field Values**

Once we begin to enter new bugs, we‘ll see a number of drop-down lists containing default values. Some of these may work just fine for our product. Others may not. We can modify the values of these fields, adding new values and deleting old ones. Let‘s take a look at the OS category.

To modify default field values:

1. At the bottom of the page, in the **Edit** section, click **Field Values**.
2. Click the link, in this case **OS**, for the field we want to edit. The OS field contains a list of operating system names. We are going to add browsers to this list. In reality, we might create a custom field instead, but for the sake of this example, just add them to the OS list.
3. Click **Add a value**. In the **Value** field, enter ―IE7.‖ Click **Add**.
4. Click **Add a value** again.
5. In the **Value** field, enter ―Firefox 3.‖
6. Click **Add**.
7. Where it reads **Add other values for the op_sys field**, click **op_sys**.
8. This redisplays the table. We should now see the two new entries at the top of the table. These values will also appear in the OS drop-down list when we create a new bug.

**Creating a New Bug**

Creating bugs is a big part of what Bugzilla does best.

To create a new bug:

1. In the top menu, click **New**.

2. If we've defined more than one component, choose the component from the component list.
3. Select a **Severity** and a **Priority**. **Severity** is self-explanatory, but **Priority** is generally assumed to be the lower the number, the higher the priority. So, a **P1** is the highest priority bug, a *showstopper*.
4. Click the **OS** drop-down list to see the options, including the new browser names we entered.
5. Select one of the options.
6. Enter a summary and a description. We can add any other information of choice, but it is not required by the system, although we may determine that our bug reporting policy requires certain information.
7. Click **Commit**. Bugzilla adds our bug report to the database and displays the detail page for that bug.

**Viewing Bug Reports**

Eventually, we'll end up with thousands of bugs listed in the system. There are several ways to view the bugs. The easiest is to click the My Bugs link at the bottom of the page. Because we've only got one bug reported, we'll use the standard Search function.

To find a bug:

1. Click **Reports**.
2. Click the **Search** link on the page, not the one in the top menu. This opens a page titled ―Find a Specific Bug.‖
3. Select the **Status**.
4. Select the **Product**.
5. Enter a word that might be in the title of the bug.
6. Click **Search**. If any bugs meet the criteria that we have entered, Bugzilla displays them in a list summary.
7. Click the **ID** number link to view the full bug report.

**Modifying Bug Reports**

Suppose we want to change the status of the bug. We've reviewed it and have determined that it belongs to one of the users we have created earlier

To modify a bug report:

1. Scroll down the full bug description and enter a comment in the **Additional Comments** field.
2. Select ―Reassign bug to‖ and replace the default user ID with one of the other user IDs you created. It must be in the format of an email address

**Experiment 9**

**Aim: Study of Any Test Management Tool ( TestDirector)**

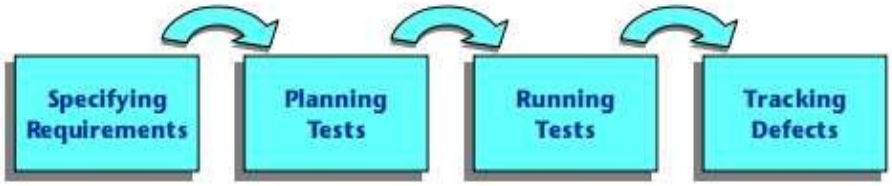
Test Director is a global test management solution which provides communication, organization, documentation and structure to the testing project.

Test Director is used for

Mapping Requirements to User acceptance test cases
Test Planning by placing all the test cases and scripts in it.
Manual testing by defining test steps and procedures
Test Execution status
Defect Management

**The TestDirector Testing Process**

TestDirector offers an organized framework for testing applications before they are deployed. Since test plans evolve with new or modified application requirements, you need a central data repository for organizing and managing the testing process. TestDirector guides through the requirements specification, test planning, test execution, and defect tracking phases of the testing process.The TestDirector testing process includes four phases:




**Specifying Requirements**

- Requirements are linked to tests and defects to provide complete traceability and aid the decision-making process
- See what percent of requirements are covered by tests

- Each requirement in the tree is described in detail, and can include any relevant attachments. The QA tester assigns the requirement a priority level which is taken into consideration when the test team creates the test plan

- Import from Microsoft Word or third party RM tool

**Planning Tests**

The Test Plan Manager enables to divide application according to functionality. Application can be divided into units, or subjects, by creating a test plan tree.

Define subjects according to:

- Application functionality-such as editing, file operations, and reporting
- Type of testing-such as functional, user interface, performance, and load

As the tests are also linked to defects, this helps ensure compliance with testing requirements throughout the testing process

**Running Tests**

As the application constantly changes, using test lab, run manual and automated tests in the project in order to locate defects and assess quality.

By creating test sets and choosing which tests to include in each set, test suite can be

created. A test set is a group of tests in a TestDirector project database designed to achieve specific testing goals.
Tests can be run manually or scheduled to run automatically based on application dependencies.

**Tracking Defects**

Locating and repairing application defects efficiently is essential to the testing process. Defects can be detected and added during all stages of the testing process. In this phase you perform the following tasks:

This tool features a sophisticated mechanism for tracking software defects, enabling Testing Team and the project Team to monitor defects closely from initial detection until resolution

- By linking TestDirector to e-mail system, defect tracking information can be shared by all Development and Management Teams, Testing and Wipro Software Quality Assurance personnel

**System Requirements for TestDirector**

Server System configuration : 128 MB of RAM , 500 MB of free disk space, Win NT server, Win 2K server, IIS 5.0, MSAccess/Oracle 7.x,8.x,9/MS SQL Server Client System configuration : 64 MB of RAM , 10 MB of free disk space, Win 95/98/NT/2K/XP, IE 5 , Netscape 4.7

**Experiment 10**

**Aim: Study of any open source testing tool (TestLink)**

Testlink is an open source test management tool. It enables creation and organization of test cases and helps manage into test plan. Allows execution of test cases from test link itself. One can easily track test results dynamically, generate reports, generate test metrics,prioritize test cases and assign unfinished tasks. Its a web based tool with GUI, which provides an ease to develop test cases, organize test cases into test plans, execute these test cases and generate reports. Test link exposes API, written in PHP, can help generate quality assurance dashboards. The functions like AddTestCase ToTestPlan, Assign Requirements,Create TestCase etc. helps create and organize test cases per test plan. Functions like GetTestCasesForTestPlan, GetLastExecutionResult allows one to create quality assurance dashboard.

TestLink enables easily to create and manage Test cases as well as organize them into Test plans. These Test plans allow team members to execute Test cases and track test results dynamically, generate reports, trace software requirements, prioritize and assign tasks. Read more about implemented features and try demo pages.

**Overall structure**

There are three cornerstones: **Product**, or attributes for this base. First, definition of documentation.

**Products and Test Plans**

**Test Plan** and **User**. All other data are relations a couple of terms that are used throughout the

*Product*: A Product is something that will exist forever in TestLink. Products will undergo many different versions throughout their life times. Product includes Test Specification

with Test Cases and should be sorted via Keywords.

*Test Plan*: Test Plans are created when you'd like to execute test cases. Test plans can be made up of the test cases of one or many Products. Test Plan includes Builds, Test Case Suite and Test Results.

*User:* An User has a Role, that defines available TestLink features.

**Test Case Categorization**

TestLink breaks down the test case structure into three levels Components, Categories, and test cases. These levels are persisted throughout the application.

*Component*: Components are the parents of Categories. Each Component can have many Categories.

*Category*: Categories are the parents of test cases. Each Category can have many test cases.

*Test Case*: Test cases are the fundamental piece of TestLink.

*Test Specification*: All Components, Categories and test cases within Product.

*Test Case Suite*: All Components, Categories and test cases within Test Plan.

**Test Specification**

**Creating Test Cases**

Tester must follow this structure: Component, Category and test case. At first you create Component(s) for your Product. Component includes Categories. Category has the similar meaning but is second level of Test Specification and includes just Test Cases.

User can also copy or move Test Cases.

Test Cases has following parts:

- Title: could include either short description or abbreviation (e.g. TL-USER-LOGIN)
- Summary: should be really short; just for overview.
- Steps: describe test scenario (input actions); can also include precondition and cleanup information here.
- Expected results: describe checkpoints and expected behaviour a tested Product or system.

**Deleting Test Cases**

Test cases, Categories, and Components may be deleted from a test plan by users with lead permissions from the ―delete test cases‖ screen. Deleting data may be useful when first creating a test plan since there are no results. However, Deleting test cases will cause the loss of all results associated with them. Therefore, extreme caution is recommended when using this functionality.

**Requirements relation**

Test cases could be related with software/system requirements as n to n. The

functionality must be enabled for a Product. User can assign Test Cases and Requirements via link Assign Requirements in the main screen.

**Test Plans**

Test plan contains name, description, collection a chosen test cases, builds, test results, milestones, tester assignment and priority definition.

**Creating a new Test Plan**

Test Plans may be deleted from the ―Create test plan‖ page (link ―Create Test Plan‖) by users with lead privileges. Test plans are the basis for test case execution. Test plans are made up of test cases imported from Products at a specific point of time. Test plans can only be created by users with lead privileges. Test plans may be created from other test plans. This allows users to create test plans from test cases that at a desired point in time. This may be necessary when creating a test plan for a patch. In order for a user to see a test plan they must have the propper rights. Rights may be assigned (by leads) in the define User/Project Rights section. This is an important thing to remember when users tell you they can‘t see the project they are working on.

**Test Execution**

Test execution is available when:

1. A Test Specification is written.

2. A Test Plan is created.

3. Test Case Suite (for the Test Plan) is defined.

4. A Build is created.

5. The Test plan is assigned to testers (otherwise they cannot navigate to this Test Plan).

Select a required Test Plan in main page and navigate to the _Execute tests‘ link. Left pane serves for navigation in Test Case Suite via tree menu, filtering and define a tested build.

**Test Status**

Execution is the process of assigning a result (pass, fail, blocked) to a test case for a specific build. _Blocked‘ test case is not possible to test for some reason (e.g. a problem in configuration disallows to run a tested functionality).

**Insert Test results**

Test Results screen is shown via click on an appropriate Component, Category or test case in navigation pane. The title shows the current build and owner. The colored bar indicate status of the test case. Yellow box includes test scenario of the test case.

**Updated Test Cases**: If users have the proper rights they can go to the ―Update modified test case‖ page through the link on main page. It is not necessary for users to update test cases if there has been a change (newer version or deleted).

**Advantages:**

1. Easy in tracking test cases(search with keyword, test case id, version etc)
2. We can add our custom fields to test cases.
3. Allocating the work either test case creation/execution any kind of documents is easy
4. when a test cases is updated the previous version also can be tracked
5. We can generate results build wise
6. Test plans are created for builds and work allocations can be done.
7. Report, is one of the awesome functionality present in the Test link, it generates reports in desired format like HTML/ CSV /Excel and we can create graphs too.
8. And the above all is done on the privileges based which is an art of the testlink and i liked this feature much

**Example of TestLink workflow:**

1. Administrator create a Product ―Fast Food‖ and a user Adam with rights ―leader‖ and Bela with rights ―Senior tester‖.

2. Adam imports Software Requirements and for part of these requirements generates empty Test cases.

3. Bela describe test sneario of these Test cases that are organized according to Components and Categories.

4. Adam creates Keyword: ―Regression‖ and assignes this keyword to ten of these test cases.

5. Adam creates a Test Plan ―Fish & Chips‖, Build ―Fish 0.1‖ and add Test Cases with keywords ―Regression‖.

6. Adam and Bela execute and record the testing with result: 5 passed, 1 failed and 4 are blocked.

7. Developers make a new build ―Fish 0.2‖ and Bela tests the failed and blocked test cases only. Exceptionaly all these five Test cases passed.

8. Manager would like to see results. Administrator explains him that he can create account himself on the login page. Manager does it. He has ―Guest‖ rights and could see results and Test cases. He can see that everything passed in overall report and problems in build ―Fish 0.1‖ in a report for particular Build. But he can change nothing.

## INTRODUCTION ABOUT LAB

**Testing** is a process used to help identify the correctness, completeness and quality of developed computer software. With that in mind, testing can never completely establish the correctness of computer software. here are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following rote procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are things the tester tries to do with the product, and the product answers with its behavior in reaction to the probing of the tester. Although most of the intellectual processes of testing are nearly identical to that of review or

inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces.

**Testing helps is verifying and Validating if the Software is working as it is intended to be working. This involves using Static and Dynamic methodologies to Test the application**

## REFERENCES


1. Testing Computer Software, 2nd Edition by Cem Kaner, Jack Falk and Hung
2. Effective Software Testing Methodology by Willian E.Perry
3. Software Testing Foundations: A Study Guide for the Certified Tester Exam (Rockynook Computing) by Andreas Spillner, Tilo Linz and Hans Schaefe.
4. Software Testing: A Craftsman's Approach, Third Edition by Paul Jorgensen


# VIVA QUESTIONS:

**1. Define SQA?**

SQA stands for Software Quality Assurance. This is the measure of assuring the quality of the software products. The major activity done here is testing. The assurance process also follows the quality model called the QAIMODEL (Quality Assurance Institute Model).

**2. What is V Testing?**

'V' testing stands for Verification and Validation testing.

**3. What is a quality?**

Quality refers to the ability of products to meet the user's needs and expectations.

**4. Name the two issues for software quality.**

Validation or user satisfaction, and verification or quality assurance.

**5. Define user satisfaction testing.**

User satisfaction testing is the process of quantifying the usability test with some measurable attributes of the test, such as functionality, cost or ease of use.

**6. Define test plan.**

A test plan is developed to detect and identify potential problems before delivering the software to its users.

**7. Write the objectives of testing.**

Testing is the process of executing a program with the intent of finding errors.

A good test case is the one that has a high probability of detecting an as yet undiscovered error.

A successful test case is the one that detects an as yet undiscovered error.

**8. What is cyclomatic complexity?**

Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program. The value computed for cyclomatic complexity defines the number of independent paths in the basis set of program.

**9. Define corollary?**

Corollary is a proposition that follows from an axiom or another proposition that has been proven.

Name the two axioms.

Axiom1: The independence axiom. Maintain the independence of components.

Axiom2: The information axiom. Minimize the information content of the design.

**10. Define coupling.**

Coupling is a measure of the strength of association established by a connection from one object or software component to another. Coupling is a binary relationship. Coupling deals with interactions between objects or software components.

**11. Name the two types of coupling in the object oriented design.**

Interaction coupling and inheritance coupling.

**12. Define cohesion.**

Cohesion means the interactions within a single object or software component.

**13. Name the types of attributes.**

Single value attribute, Multiplicity or multi-value attributes, Reference to another object or instance connection.

**14. Write the syntax for presenting the attribute that was suggested by UML.**

visibility name : type_expression = initial _value

Where visibility is one of the following

+ public visibility

# protected visibility

- private visibility

type_expression - type of an attribute

Initial_value is a language dependent expression for the initial value of a newly created object.

**15.Write the syntax for presenting the operation that was suggested by UML**

Visibility name: (parameter_list): return _type_expression

Where visibility is one of the following

+ public visibility

# protected visibility

- private visibility

parameter- is a list of parameters.

Return_type_expression: is a language _dependent specification of the

Implementation of the value returned by the method.

**16. What is a Façade?**

Facade classes are the public classes in a package for public behavior.

**17. Define DBMS?**

A database management system (DBMS) is a program that enables the creation and maintenance of a collection of related data.

**18. What is database model?**

Database model is a collection of logical constructs used to represent the data structure and data relationships within the database.

**19. Name the two categories of database model?**

Conceptual model and Implementation model.

**20. Write the six categories for the life time of data**

Transient results to the evaluation of expressions, variables involves in procedure activation, global variables and variables that are dynamically allocated, data that exist between the execution of a program, data that exist between the versions of a program, data that outlive a program.

**21. What is schema or metadata?**

The fundamental characteristic of the database is that the DBMS contains not only the data but the complete definition of the data formats such as data structures, types and constraints, it manages. This description is known as the schema or metadata

**22. Name the three types of data base model?**

Hierarchical model, network model, relational model.

**23. Define data definition language.**

Data definition language (DDL) is a language used to describe the structure of and relationships between objects stored in a database .This structure of information are termed as database schema.

**24. Define data manipulation language.**

Data manipulation language (DML) is a language that allows users to access and manipulate (such as create, save, or destroy) data organization.

**25. When the transaction is said to commit.**

The transaction is said to commit if all changes can be made successfully to the database.

**26. When the transaction is said to abort.**

The transaction is said to abort if all changes to the database can not be made successfully.

**27. What is conservative or pessimistic policy?**

The most conservative way to enforce serialization is to allow a user to lock all objects or records when they are accessed and to release the locks only after a transaction commits. This approach is known as conservative or pessimistic policy.

**28. Describe client server computing.**

The client is a process (program) that sends a message to a server process (program) requesting that the server perform a task (service).

**29. Name the types of object relation mapping.**

Table class mapping, Table –multiple classes mapping, Table-Inherited classes mapping, Tables-Inherited classes mapping.

**30. Write the need of middleware.**

The client is a process (program) that sends a message to a server process (program) requesting that the server perform a task (service). The key element of connectivity is the network operating system (NOS), also known as middleware.

**31. Mention the different forms of server.**

File server, database server, transaction server, application server.

**32. What is the use of application web server?**

In a two-tier architecture, a client talks directly to a server, no intervening server. Three_ tier architecture introduces a server that is application web server between the client and the server to send and receive the messages.

**33. Write the components of client server application.**

User interface, business processing, database processing.

**34. What is Object Oriented Database management system?**

Object Oriented Database management system is a marriage of Object Oriented programming and Database management system.

**35. Define ODBC?**

The Open Database connectivity is an application programming interface that provides solutions to the multi database programming interface.

**36. What is the need of an Interaction diagram?**

An Interaction diagram is used to trace the exception of a scenario in the same context of an object diagram.

**37. What is the need of a Class diagram?**

A class diagram is used to show the existence of classes and their relationships in the logical view of a system.

**38. What is Behavior of an object?**

Behavior is how an object acts and reacts in terms of its state changes and message passing.

**39. What are the characteristic features of an Interaction diagram?**

They include the representation of objects with its name and class name. Each object has a life line. The order of messaging between objects is well defined.

**40. Define forward engineering and revere engineering.**

Forward engineering means creating a relational schema from an existing object model

Reverse engineering means creating an object model from an existing relational database layout (schema).

**41. What is Object request broker (ORB)?**

Object request broker (ORB) –Middle ware that implements a communication channel through which applications can access object interfaces and request data and services.

**42. What is distributed database?**

In distributed database, different portions of the database reside on different nodes (computers) and disk drives in the network. Each portions of the database is managed by a server, a process responsible for controlling access and retrieval of data from the database portion.

**43. What does RAD stands for?**

Rapid application development (RAD) is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods.

**44. What are the traditional software development methodologies?**

Most traditional development methodologies are either algorithm centric or data centric.

**45. What is the MAIN benefit of designing tests early in the life cycle?**

It helps prevent defects from being introduced into the code.

**46. What is risk-based testing?**

Risk-based testing is the term used for an approach to creating a test strategy that is based on prioritizing tests by risk. The basis of the approach is a detailed risk analysis and prioritizing of risks by risk level. Tests to address each risk are then specified, starting with the highest risk first.

**47. A wholesaler sells printer cartridges. The minimum order quantity is 5. There is a 20% discount for orders of 100 or more printer cartridges. You have been asked to prepare test cases using various values for the number of printer cartridges ordered. Which of the following groups contain three test inputs that would be generated using Boundary Value Analysis?**

4, 5, 99

**48. What is the KEY difference between preventative and reactive approaches to testing?**

Preventative tests are designed early; reactive tests are designed after the software has been produced.

**49. What is the purpose of exit criteria?**

To define when a test level is complete.

**50. What determines the level of risk?**

The likelihood of an adverse event and the impact of the event

**51. When is used Decision table testing?**

Decision table testing is used for testing systems for which the specification takes the form of rules or cause-effect combinations. In a decision table the inputs are listed in a column, with the outputs in the same column but below the inputs. The remainder of the table explores combinations of inputs to define the outputs produced.

Learn More about Decision Table Testing Technique in the Video Tutorial here

**52. What is the MAIN objective when reviewing a software deliverable?**

To identify defects in any software work product.

**53. Which of the following defines the expected results of a test? Test case specification or test design specification.**

Test case specification.

**54. Which is a benefit of test independence?**

It avoids author bias in defining effective tests.

**55. As part of which test process do you determine the exit criteria?**

Test planning.

**56. What is beta testing?**

Testing performed by potential customers at their own locations.