\Software Testing

LAB MANUAL

Contents:

Vision & Mission

Vision of the Institute

Confect as a premier institute for professional education by creating technocrats who can address the society`s needs through inventions and innovations.

Mission of the Institute

- Partake in the national growth of technological, industrial arena with societal responsibilities.
- Provide an environment that promotes productive research.
- Meet stakeholders expectations through continued and sustained quality improvements.

Vision of the Program

To create Technical & Managerial expertise, ethically strong global manpower by providing top class Information Technology education to deliver the needs of the society to excel in real life situation at all levels.

Mission of the Program

To provide best supporting learning environment and development of knowledge well trained, confident, industry ready professionals and an inquisitive mind, who are ready to contribute to the industry, economy & the society

PEOs & POs

Program Educational Objectives

This education is meant to prepare our students to thrive and to lead. In their careers, our graduates will be able

P1	To develop the ability among the graduates to gain knowledge about core and Application domain subjects in Information Technology
P2	To prepare the graduates for successful careers in industry and make them compatible Information Technology professionals to meet the global needs.
P3	To promote graduates awareness for life-long learning and to introduce them to professional ethics.

Program Outcomes

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

1. **Mobile & Web Application Development**: Ability to develop mobile & web applications using J2EE, Android and J2ME.
2. **Cloud Services**: To deploy virtualized and cloud based services in the organization

Preface:

This manual is intended for the Third year students of B. Tech Information Technology in the subject of Software Testing. This manual typically contains practical/Lab Sessions related Software Testing covering various aspects related the subject to enhanced understanding.

Although, as per the syllabus, study of Test cases is prescribed, we have made the efforts to cover various aspects of Software Testing covering different testing types

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

The Software Engineering lab will facilitate the students to develop a preliminary yet practical understanding of software development process and tools

Authors

LAB OBJECTIVES:

This course will enable the students to

- Demonstrate the UML diagrams with ATM system descriptions.
- Demonstrate the working of software testing tools with c language.
- Study of testing tools- wins runner, selenium etc.
- Writing test cases for various applications

LAB OUTCOMES :

After the completion of this course student will be able to

- Find practical solutions to the problems
- Solve specific problems alone or in teams
- Manage a project from beginning to end
- Work independently as well as in teams
- Define, formulate and analyze a problem

RECOMMENDED SYSTEMS / SOFTWARE REQUIREMENTS :

- Intel based desktop PC with P-IV or above, minimum of 166MHZ or faster processor with at least 512MB RAM and 10GB free disk space.
- Windows OS with MS Office Package. Recommended.

Guidelines to the Students

- Be regular to the Lab. After entering into the Lab, Sign in the Log-register.
- Students should come with observation and record.
- Lab attendance will play a part of the internal assessment marks
- The students should come to the lab with a prior preparation to complete the programs within time.
- Students should get their observation books corrected before leaving the lab and should submit the records in next lab session.
- Follow the dress-code while coming to the Lab. Strict discipline should be maintained inside the laboratory.
- At the mid and end of the semester, lab exams will be conducted.
- Shut down the system properly, turn-off the monitor, Arrange the chair properly and then leave. Handle the systems carefully.

JAHAWARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

Academic Year 2018-19

III Year B.Tech–II Sem.

SOFTWARE TESTING LAB

List of experiments as per the university

No. of Credits: 2

PROGRAMS LIST:

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).
   d) How to synchronize a test when the application responds slowly.
   e) How to create a test that checks GUI objects and compare the behaviour of GUI objects in different versions of the sample application.
   f) 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.
   g) How to Create Data-Driven Tests which supports to run a single test on several sets of data from a data table.
   h) How to read and check text found in GUI objects and bitmaps.
   i) How to create a batch test that automatically runs the tests.
   j) How to update the GUI object descriptions which in turn supports test scripts as the application changes.
7. Apply Win Runner testing tool implementation in any real time applications

Academic Year 2018-19
III Year B.Tech–II Sem.
SOFTWARE TESTING LAB
List of experiments to be conducted for this semester

No. of Credits: 2

Experiment-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

Experiment-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.

Experiment-3:

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

Experiment-4:

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

Experiment-5:

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

Experiment-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).

Experiment-7:

a) How to synchronize a test when the application responds slowly.
b) How to create a test that checks GUI objects and compare the behavior of GUI objects in different versions of the sample application.
c) 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.

Experiment-8:

a) How to Create Data-Driven Tests which supports to run a single test on several sets of data from a data table.
b) How to read and check text found in GUI objects and bitmaps.

Experiment-9:

a) How to create a batch test that automatically runs the tests.
b) How to update the GUI object descriptions which in turn supports test scripts as the application changes.

Experiment-10:

Apply Win Runner testing tool implementation in any real time applications

Mini Projects:

1. Study of any test management tool (e.g. Test Director
2. Study of any bug tracking tool (e.g. Bugzilla, bugbit)

Index

Prog No	Date of Completion	Name of Program/ Experiment	Marks/ Grade	Signature of Faculty

Software Testing

Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item. Software testing is an activity that should be done throughout the whole development process.

Software testing is a process of the "verification and validation," or V&V, software practices. Some other V&V practices, such as inspections and pair programming, will be discussed throughout this book. Verification (the first V) is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Verification activities include testing and reviews. For example, in the software for the Monopoly game, we can verify that two players cannot own the same house. Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. At the end of development validation (the second V) activities are used to evaluate whether the features that have been built into the software satisfy the customer requirements and are traceable to customer requirements. Verification and validation as follows: Actual

Verification: Are we building the product right?

"I landed on "Go" but didn't get my $"

Validation: Are we building the right product?

"I know this game "Go" – but this is not the game I wanted.

Terminology :

- **Mistake** – a human action that produces an incorrect result.
- **Fault [or Defect]** – an incorrect step, process, or data definition in a program.

- **Failure** – the inability of a system or component to perform its required function within the specified performance requirement.
- **Error** – the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
- **Specification** – a document that specifies in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristic of a system or component, and often the procedures for determining whether these provisions have been satisfied.

Types of Testing:

- **Black box testing** (also called functional testing) is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.
- **White box testing** (also called structural testing and glass box testing) is testing that takes into account the internal mechanism of a system or component.

1. Unit Testing

Opacity: White box testing

Specification: Low-level design and/or code structure

Unit testing is the testing of individual hardware or software units or groups of related units. Using white box testing techniques, testers (usually the developers creating the code implementation) verify that the code does what it is intended to do at a very low structural level. For example, the tester will write some test code that will call a method with certain parameters and will ensure that the return value of this method is as expected. Looking at the code itself, the tester might notice that there is a branch (an if-then) and might write a second test case to go down the path not executed by the first test case. When available, the tester will examine the low-level design of the code; otherwise, the tester will examine the structure of the code by looking at the code itself. Unit testing is generally done within a class or a component.

Integration testing :

Opacity: Black- and white-box testing

Specification: Low- and high-level design

Integration test is testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them .Using both black and white box testing techniques, the tester (still usually the software developer) verifies

that units work together when they are integrated into a larger code base. Just because the components work individually, that doesn't mean that they all work together when assembled or integrated. For example, data might get lost across an interface, messages might not get passed properly, or interfaces might not be implemented as specified. To plan these integration test cases, testers look at high- and low-level design documents.

3. **Functional and system testing**

Opacity: Black-box testing

Specification: high-level design, requirements specification

Using black box testing techniques, testers examine the high-level design and the customer requirements specification to plan the test cases to ensure the code does what it is intended to do. ***Functional testing*** involves ensuring that the functionality specified in the requirement specification works. System testing involves putting the new program in many different environments to ensure the program works in typical customer environments with various versions and types of operating systems and/or applications.

System **testing** is testing conducted on a complete, integrated system to evaluate the system compliance with its specified requirements. Because system test is done with a full system implementation and environment, several classes of testing can be done that can examine non-functional properties of the system. It is best when function and system testing is done by an unbiased, independent perspective (e.g. not the programmer).

Stress testing – testing conducted to evaluate a system or component at or beyond the limits of its specification or requirement.

Performance testing – testing conducted to evaluate the compliance of a system or component with specified performance requirements [11]. To continue the above example, a performance requirement might state that the price lookup must complete in less than 1 second. Performance testing evaluates whether the system can look up prices in less than 1 second (even if there are 30 cash registers running simultaneously).

Usability testing – testing conducted to evaluate the extent to which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. While stress and usability testing can be and is often automated, usability testing is done by human-computer interaction specialists that observe humans interacting with the system.

4. Acceptance testing

Opacity: Black-box testing

Specification: requirements specification

After functional and system testing, the product is delivered to a customer and the customer runs black box acceptance tests based on their expectations of the functionality. Acceptance testing is formal testing conducted to determine whether or not a system satisfies its acceptance criteria (the criteria the system must satisfy to be accepted by a customer) and to enable the customer to determine whether or not to accept the system. These tests are often pre-specified by the customer and given to the test team to run before attempting to deliver the product. The customer reserves the right to refuse delivery of the software if the acceptance test cases do not pass.

5. Regression testing

Opacity: Black- and white-box testing

Specification: Any changed documentation, high-level design throughout all testing cycles, *regression* test cases are run. Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements. Regression tests are a subset of the original set of test cases. These test cases are re-run often, after any significant changes (bug fixes or enhancements) are made to the code. The purpose of running the regression test case is to make a "spot check" to examine whether the new code works properly and has not damaged any Previously-working functionality by propagating unintended side effects. Most often, it is impractical to re-run all the test cases when changes are made. Since regression tests are run throughout the development cycle, there can be white box regression tests

6. Beta testing

Opacity: Black-box testing

Specification: None.

When an advanced partial or full version of a software package is available, the development organization can offer it free to one or more (and sometimes thousands) potential users or *beta testers*. These users install the software and use it as they wish, with the understanding that they will report any errors revealed during usage back to the

development organization. These users are usually chosen because they are experienced users of prior versions or competitive products.

The advantages of running beta tests are as follows

- *Identification of unexpected errors* because the beta testers use the software in unexpected ways.
- *A wider population search for errors* in a variety of environments (different operating systems with a variety of service releases and with a multitude of other applications.
- *Low costs* because the beta testers generally get free software but are not compensated.
- *Lack of systematic testing* because each user uses the product in any manner they choose.
- *Low quality error reports* because the users may not actually report errors or may report errors without enough detail.
- *Much effort is necessary to examine error reports* particularly when there are many beta testers.

Testing Type	Specification	General Scope	Opacity	Who generally does it?
Unit	Low-Level Design Actual Code Structure	Small unit of code no larger than a class	White Box	Programmer who wrote code
Integration	Low-Level Design High-Level Design	Multiple classes	White Box Black Box	Programmers who wrote code
Functional	High Level Design	Whole product	Black Box	Independent tester
System	Requirements Analysis	Whole product in representative environments	Black Box	Independent tester
Acceptance	Requirements Analysis	Whole product in customer's environment	Black Box	Customer
Beta	Ad hoc	Whole product in customer's environment	Black box	Customer
Regression	Changed Documentation High-Level Design	Any of the above	Black Box White Box	Programmer(s) or independent testers

Table 1: Levels of Software Testing

The Test Case: Test Cases are the set of conditions or variables for checking this. For each scenario there will be test cases, and these set of conditions are planned by the tester.The format of your test case is given below

Test ID	**Description**	**Expected Results**	**Actual Results**

VARIOUS FILEDS IN A TEST CASE

TEST CASE FIELD	DESCRIPTION
Test case ID:	Each test case should be represented by a unique ID. To indicate test types follow some convention like "TC UI 1" indicating "User Interface Test Case#1."
Test Priority:	It is useful while executing the test. o Low o Medium o High
Name of the Module:	Determine the name of the main module or sub-module being tested
Test Designed by:	Tester's Name
Date of test designed:	Date when test was designed
Test Executed by:	Who executed the test- tester
Date of the Test Execution:	Date when test needs to be executed
Name or Test Title:	Title of the test case
Description/Summary of Test:	Determine the summary or test purpose in brief
Pre-condition:	Any requirement that needs to be done before execution of this test case. To execute this test case list all pre-conditions
Dependencies:	Determine any dependencies on test requirements or other test cases
Test Steps:	Mention all the test steps in detail and write in the order in which it requires to be executed. While writing test steps ensure that you provide as much detail as you can
Test Data:	Use of test data as an input for the test case. Deliver different data sets with precise values to be used as an input
Expected Results:	Mention the expected result including error or message that should appear on screen
Post-Condition:	What would be the state of the system after running the test case?
Actual Result:	After test execution, actual test result should be filled
Status (Fail/Pass):	Mark this field as failed, if actual result is not as per the estimated result

WinRunner

- It's a Functionality Testing Tool.
- It's a Mercury Interactive's Testing tool.
- Supports VB, VC++, Java, Power Builder, Delphi, D2K, HTML, Siebel.
- It Records GUI operations in Record mode.
- While Recording, it Automatically generates a Test Script Language(TSL).
- It can do functional Testing of a variety of application software written in programming languages such as PowerBuilder, VB, C/C++, JAVA and also on ERP/CRM software packages.
- Run on Windows family Operating Systems.
- XRunner works on Unix and Linux Platforms.
- It Perform Testing in all Windows OS, Browser Environments such as Internet Explorer.
- It can add checkpoints to compare actual and expected results.
- It provides a facility for synchronization of Test Cases.
- An automated program to apply a test on Application build is called a Test Script.
- It supports Auto-learning to Recognize Objects and Windows during recording.
- It support two types of modes of recording.
- Context Sensitive Mode
- Analog Mode
- In the 1st mode, it records *Mouse & Keyboard* operations w.r.t. to objects and windows.
- In the 2nd mode, it records *Mouse pointer* movements on the desktop.
- To change into analog mode follow the steps below.
- Click *"Start Recording"* twice.
- Go to *Create* Menu and then click *Record Analog*
- Press *F2* to change from one mode to other.
- Analog mode is used to record digital signatures, Graph drawing and Image movements.

Testing Process

- Learning (Objects/Windows Recognition)
- Recording (Manual Test into an Automation)
- Edit Script (Checkpoints & Ctrl Stmts)
- Run Script (Execution of Automated Script)
- Analyze Test Results (Defect Tracking for Test Engineers)

Testing an Application

- After installing the software, invoke the WinRunner application
- Start → All Programs → WinRunner →WinRunner
- Give Checkmarks for all Add-Ins in the boxes and click Ok.
- Select "New test" to Create a New Test Script (or)
- "Open test" to Open an Existing Test Script (or)
- "Quick Preview" to View the Quick Preview of WinRunner

Recording Testcases

- To test any application first run it and understand its operation.

- Invoke WinRunner,again run the application and record the GUI operations.
- During recording mode,it captures all your actions.
- Once the recording is completed, WinRunner generates a script in TSL.
- Run this Test Script to View the Results generated by WinRunner.
- Test Results will show,If the test has Passed or Failed
- There are Two Modes of recording

  1.Context Sensitive Mode (CSM)

  2.Analog Mode (AM)
- CSM is used If the locations of GUI Controls or the mouse positions are not necessary
- If the Mouse Positions and Locations of Controls are needed then use AM ,used to validate bitmaps,testing the signatures etc.

Procedure for Recording a Testcase

- 1.File → New [or select "new test "from Welcome Screen]
- 2.Open [run] → The application to be tested
- 3.Start recording a test case
- Create → Record Context Sensitive [Click "Record" once]
- 4.Select the application to be tested by clicking on the application's title bar
- 5.Perform all the actions to be recorded
- 6.Once all required actions are recorded, stop the recording

In GUI Map Editor ,WinRunner automatically saves the information from the application

Procedure to save the information:

- Tools → GUI MapEditor → file → save as → Enter File Name
- →Close GUI Map Editor

Procedure for loading the GUI Map File:

- Tools → GUI Map Editor → File → Close all → File → Open → Select File (x.gui) → list of items displayed during recording phase (double click on application)

Procedure for Running a Test Case:

- Open a test script to be executed →run the test (Run →Run from Top or F5)
- WinRunner executes the generated script and displays the results in "Test Result Window"

Testing Standard Calculator Application Using WinRunner

- Invoke the calculator application from start →Programs →Accessories →Calculator
- To test complete functionality of the application ,first Generate Test Cases ,which covers all buttons in it

Prepare Test Cases like Some of testcases with correct outputs and some of the testcases with error messages.

Test cases and Expected outputs for Testing the Calculator Application

Test Case	Expected Output
1. 4, 1/x	0.25
2.-6 , sqrt	"Invalid i/p for function"
3. 4 , C	Clears the display
4. 1.2 * 3	3.6
5. 5/2.0	2.5
6. 7 + 8 – 9	6
7. 600 * 2 %	12
8. 2,MS,C,MR	2
9.MC,2,M+,3,M+,C,MR	5

Test case #1: To test Inverse operation (Inverse of 4 using /x button)
Step 1 : Open WinRunner application
Step 2 : Open Calculator application
Step 3 : Create a newdocument [File →new] (or) CTRL+N
Step 4 : Start Recording[Click "record" button]
Step 5 : Select Calculator application and start recording the actions(Click 4 → 1/x → 0.25)
Step 6 : "Stop" recording [Click "stop" button]
Step 7 : Once the recording is over, WinRunner generates TSL script
Step 8 : Save the file "Inverse" in selected folder
Step 9 : Run from Top (F5)
Step 10: After executing TSL statements ,WinRunner generates test results with the details like Passed/failed ,line no ,events ,Details ,Result ,Time, Name of the test case etc.

Test Case #2 : To test the operation ("Square root of -6")
Test Case #3 : To clear the display after performing some operations(4 ,C)
Test Case #4 : Multiplication of 2 numbers(1.2*3)
Test Case #5 : Division of 2 numbers (5/2.0)
Test Case #6 : To test the operation(7+8-9)
Test Case #7: To test the operation 2 % of 600 (600*%2)
Test Case #8 : To test MS and MR buttons (2,MS,C,MR)
Test Case #9 : To test M+ and MR buttons (MC,2,M+,3,M+,C,MR)
Test Case #10 : To test MC button
Test Case #11 : To test Backspace.

WinRunner provides 4 types of check points.

- GUI Checkpoint
- Bitmap Checkpoint
- Database Checkpoint
- Text Checkpoint

GUI Checkpoint used to Verify Properties Of objects which has 3 options such as

- For Single Property (Verify a single property of an object)
- For Object/Window (Verify more than 1 property of a Single Object)
- For Multiple Objects (Verify multiple objects with multiple properties)

Navigation/Procedure

- Select Position in the Script
- Go to Create -> GUI Checkpoint
- For Single Property/For Object/Window/For Multiple Objects
- Select Testable Object(s)
- Select Property(s) with Expected Value(s) -> Paste
- It performs Changes in check list files and expected value files due to sudden changes in Req's/Test Engineer mistakes.
- Changes in expected values - due to sudden changes in Req's/Test Engineer mistakes they can change in expected values through below navigation.
    1. Execute Existing Script
    2. Open Test Results and Perform Changes in Expected Values
    3. Click Ok and Re-Execute the Test to get Correct Result.
- Add new property – we can add new properties to the existing checkpoints through below navigation.
    1. Create -> Edit GUI Checklist -> Select Checklist Filename
    2. Click Ok -> Select New Properties to add
    3. Click Ok -> Click Ok to Overwrite -> Click Ok after reading suggestion
    4. Change the run Mode "Verify" to "Update"
    5. Click Run in Verify mode to get results
    6. Analyze the results.

Bitmap Checkpoints

- Its Used to Compare the Images.
  Eg: Logo Testing, Digital Signatures/Graphs Comparison.
- It consists of two sub options
    - ❖ For Object/Window – Compares Expected Image Vs Actual.
    - ❖ For Screen Area - Compares Expected Image area with Actual.

Procedure for 1st sub option:

1. Select Position in the Script.
2. Create -> Bitmap Checkpoint -> For Object/Window
3. Select Expected Image (Double Click)

Procedure for 2nd sub option:

1. Select Position in the Script.
2. Create -> Bitmap Checkpoint -> For Screen Area
3. Select Required Image Region -> Right Click to Relieve

Database Checkpoints

- Its used to Automate Backend Testing to verify the impact of frontend operations on the content of backend tables.
- Provides Data Integrity and Data Validation

Procedure:

1. Connect to the Database
2. Execute the Select Statement
3. Capture the result in the Excel Sheet
4. Analyze the result for Data Validation and Data Integrity.

- It Consists of 3 Sub Options.
    1. Default Check – Backend Testing depends on DB Content.

2. Custom Check – Depends on Rows/Columns Count & Content.
3. Runtime Record Check – Maps B/W Frontend & Backend Columns.

Text Checkpoints

- Its used to Conduct Calculations and Other Text Based Tests we can use gettext option in Create Menu.
- It consists of 2 sub options
    1. From Object/Window – captures an object value into a variable.
    2. From Screen Area – capture the static text from screens.

Procedure for 1st sub option:

- Create -> gettext
- From Object/Window
- Select Object (Double Click).

Procedure for 2nd sub option:

- Create -> gettext
- From Screen Area -> Select Required Value Region
- Right Click to Relieve.

Data Driven Test

- Test Engineers execute tests with multiple test data to validate functionality is also known as “Retesting”.
- There are 4 types of Data Driven tests.
    1. Dynamic Test Data Submission - Execute tests with multiple test data with tester
    2. Through Flat files (.txt) - Conducting retesting based on multiple test data in flat files
    3. From Front End Grids - Conducting retesting based on multiple data objects like Lists/Menu/ActiveX/Table.

Experiment No-1 **Date: / / 20**

Aim : Write programs in 'C' Language to demonstrate the working of the following constructs:

i)do...while ii)while iii) if…else iv) switch v) for

Description:

- To understand the working of "do while" with different range of values and test cases
- To understand the working of "while" with different range of values and test cases
- To understand the working of "if else" with different range of values and test cases
- To understand the working of "switch" with different range of values and test cases
- To understand the working of "for" with different range of values and test cases

Program using do while:

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();
}
```

Output:

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		

Program using while:

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();
}
```

Output:

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		

Program using " if else" :

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();
 }
```

Output:

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		

Program using "Switch":

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 Enter a, b Values: 3, 2	The sum of a & b is:5
Enter Ur choice: 2 Enter a, b Values: 3, 2	The diff of a & b is: 1
Enter Ur choice: 3 Enter a, b Values: 3, 2	The Mul of a & b is: 6
Enter Ur choice: 4 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 Enter a, b Values: 3, 2	The sum of a & b is: 5	5	
Enter Ur choice: 2 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: Divide by zero

Input	Expected output	Actual output	Remarks
Option: 4 a= 10 & b=0	error		fail

Program using "for"

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 number	fail
	-2 is even number	-2 is odd number	

Viva Questions:

1. What is Software Testing?
2. What are the different types of testing?
3. Define the Range of values?
4. Differentiate Actual And Expected Output?

Experiment No-2 **Date: / / 20**

Introspection of failures in Matrix Multiplication

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".

Description: Understand the failures of matrix multiplication

Program:

```
#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 Matrix2 rows & cols= 3 3 Matrix1: 1 1 1 1 1 1 1 1 1 Matrix2: 1 1 1 1 1 1 1 1 1	3 3 3 3 3 3 3 3 3	3 3 3 3 3 3 3 3 3	Success

Test case no:2

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

Input	Expected output	Actual output	Remarks
Matrix1 rows & cols= 2 2 Matrix2 rows & cols= 3 2	Operation Can't be Performed		fail

Viva Questions:

1. Define pointer. Give its syntax?
2. What is double pointer?
3. Differentiate between structure and arrays?
4. What is pointer to structure?
5. What are multidimensional arrays.

Experiment No-3 **Date: / / 20**

Aim: study of ATM system specifications and report the various bugs.

The software to be designed will control a simulated automated teller machine (ATM) having a magnetic stripe reader for reading an ATM card, a customer on Solution (keyboard and display) for interaction with the customer, a slot for depositing envelopes, a dispenser for cash (in multiples of $20), a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM will communicate with the bank's computer over an appropriate communication link. (The software on the latter is not part of the requirements for this problem.)

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be
retained in the machine until the customer indicates that he/she desires no further transactions, at which point it will be returned - except as noted below. The ATM must be able to provide the following services to the customer:

- A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.
- A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM,
  subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.
- A customer must be able to make a transfer of money between any two accounts linked to the card.
- A customer must be able to make a balance inquiry of any account linked to the card.
- A customer must be able to abort a transaction in progress by pressing the Cancel key instead of responding to a request from the machine.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved. In the case of a deposit, a second message will be sent to the bank indicating that the customer has deposited the envelope. (If the customer fails to deposit the envelope within the timeout period, or presses cancel instead, no second message will be sent to the bank and the deposit will not be credited to the customer.)If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the
PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account ("to" account for transfers).

The ATM will have a key-operated switch that will allow an operator to start and stop the servicing of customers. After turning the switch to the "on" position, the operator will be required to verify and enter
the total cash on hand. The machine can only be turned off when it is not servicing a customer. When the switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc.

The ATM will also maintain an internal log of transactions to facilitate resolution using ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected) for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and dollar amounts, but for security will *never* contain a PIN.

Various Bugs in ATM System

Bug Priority:

World Web dictionary defines the word "Priority" as - Status established in order of importance or urgency. Priority of a defect/bug tells us how soon it is required to fix the problem. Priority reflects a business decision as to how soon that bug should be fixed. Priority of the bug determines what gets fixed next and what does not. The priority of a bug can be decided from either the Project management point of view or from the user's point of view. A bug that is of high priority from the Project management point of view may or may not be of high priority if assessed from the user's point of view. Check with your users; their prioritizations may surprise you! Priority of a bug can be classified as:

P1 – FIX the bug ASAP before the release, preferably in the current development iteration itself.
P2 – The Bug can be fixed in the successive iteration. But the fix must be done before the next major release of the Product.
P3 – The bug can be left to be fixed in the subsequent releases.
P4 – There is no hurry. This bug can be fixed as and when time permits.

Bug Severity Vs. Bug Priority:

The severity of a bug does not necessarily translate into the urgency to fix it. A severe bug that crashes the application only once on a Feb 29th (leap year) for 0.0001% of the users is lower priority than a mishandled error condition resulting in the need to re-enter a portion of the input for every user every time. Many bugs cause crashes, but aren't fixed because the crash is very infrequent or on a version/platform/feature low on the vendor's support list. Corner-case crashes, crashes that are dependent on a rare combination of sequence of events (trigger to set off the crash) fall under such classification of low priority bugs.

ATM Machine Example!
Let's take the example of an ATM machine to further understand the different possible combinations of bug severity and priority:

Low Severity and High Priority Bug:

Suppose you are testing an ATM machine and you notice that the welcome screen displays a misspelled Bank name! (e.g. in place of "Welcome to ICICI Bank" it rather displays "Welcome to UCICI Bank". This is quite possible, considering the fact that the letters "U" and "I" share close neighborhood in a typical QWERTY type keyboard). This might be a cosmetic error and would not hamper the functioning of the ATM machine in any possible way; still this could prove to be a real pain in the eyes for a customer. Considering that the spelling error is in a frequently used part of the program, it might give an overall bad impression that could hurt the Bank's reputation. These kinds of bugs are real annoyance for the customers and no Bank would want to loose revenue for a silly error like this one. So though this bug is of low severity, its priority would be high for obvious reasons. It is quite interesting that often, low severity cosmetic errors like these get a high priority.

High Severity and Low Priority Bug: Think of a bug that cause the ATM machine to black out if an user tries to use an expired ATM card and try it for 13 consecutive times (even after the machine keeps rejecting it)! Sounds like a critical crash? Well, it might be a critical bug as far as its severity is concerned, but don't be too surprised if you see a low Priority being assigned to it
High Severity and High Priority Bug: These kinds of bugs are easier to imagine. Any kind of bug that results in some major catastrophic failure (crashes, system hang, memory corruptions, data loss, functionalities that does not work and so on) and that occurs in areas of the application that are frequently used can be classified under such types of high severity and high priority bugs. Think of scenarios where the ATM machine - does not detect a valid card even after entering correct PIN number, hangs if tried with an invalid card and/or PIN number, blocks the card after just 1 unsuccessful entry of PIN (instead of 3 wrong entries), does not dispense money even if there is sufficient cash in the a/c and in the ATM machine, dispenses incorrect amount to the user, does not forget the previous session even after the transaction is complete (in case of ATM machines where you have to swipe your card instead of inserting it), does not return the card even after the transaction is complete and so on. As you can imagine all of these cases of bugs can fall under high severity and high priority type.

Low Severity and Low Priority Bug: These are often low impact low on urgency-meter bugs that are quite harmless and can be fixed without a hurry. These kind of bugs do not harm the application in a disastrous way and the chance of an end user/customer being annoyed by it is also very less. Think of a scenario where the ATM machine does not append a title (Mr/Miss/Mrs) to the name of the customer in the welcome screen. This would not affect the business in a drastic way.

Various Bugs:

- Machine is accepting ATM card.
- Machine is rejecting expired card.
- Successful entry of PIN number.
- Unsuccessful operation due to enter wrong PIN number 3 times.
- Successful selection of language.
- Successful selection of account type.
- Unsuccessful operation due to invalid account type.
- Successful selection of amount to be withdrawn.
- Successful withdrawal.
- Expected message due to amount is greater than day limit.
- Unsuccessful withdraw operation due to lack of money in ATM.
- Expected message due to amount to withdraw is greater than possible balance.
- Unsuccessful withdraw operation due to click cancel after insert card.

Viva Questions:

1. What are test design bugs?
2. Explain about nightmare list?
3. What are levels of testing?
4. Compare small versus large programming?
5. Define pesticide paradox?

Experiment No-4 **Date: / / 20**

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

Solution: The characteristics of a Banking application are as follows:

- Multitier functionality to support thousands of concurrent user sessions
- Large scale Integration , typically a banking application integrates with numerous other applications such as Bill Pay utility and Trading accounts
- Complex Business workflows
- Real Time and Batch processing
- High rate of Transactions per seconds
- Secure Transactions
- Robust Reporting section to keep track of day to day transactions
- Strong Auditing to troubleshoot customer issues
- Massive storage system
- Disaster Management.

File Open - Test case

Steps to reproduce:

1. Launch Application

2. Select "File" menu
File menu pulls down

3. Choose "Open"
"Open" dialog box appears

4. Select a file to open

5. Click OK

Result: File should open

Test case
Test case ID: 1
Test Description : verify B - bold formatting to the text
Function to be tested : B - bold formatting to the text
Environment : Windows
Test Execution:
1. Open program
2. Open new document
3. Type any text

4. Select the text to make bold.
5. Click Bold

Expected Result: Applies bold formatting to the text
Actual Result: pass

Sample test case for an online purchasing system

1)click the URL link for the online purchasing site.

2)Register if not registered before, Enter user name, login id and password, Click 'OK' button to submit.

3)Enter your login id and password, if you are a valid customer

4)Validate the login id and password, if correct menu page will be opened.

5) If not correct, display the error message "invalid login id or password". Re-enter your login id and password.

6)Click the item field and select the item.

7)If available, display the cost and brand, else display " item is unavailable".

8)click 'YES' to do online purchasing for the item else click 'NO'.

9)If 'YES' go to the payment type, else display the 'EXIT' page.

10)click the payment type options and select the mode of payment.

11)Fill the fields- account holder's name, bank name, credit/debit card option, card detail, price to be paid, email id etc.

12) If invalid information is given, show error message " please re-enter "

13)If black field is present, display message "all the fields are mandatory".

14)Click "SUBMIT" button.

15)Display the successful submission message "successfully completed".

16)Press 'EXIT' button.

17)Show the message " successfully logged off".

Test Cases for Banking Application

- Checking mandatory input parameters.
- Checking optional input parameters
- Check whether able to create account entity.
- Check whether you are able to deposit an amount in the newly created account (and thus updating the balance).
- Check whether you are able to withdraw an amount in the newly created account (after deposit)(and thus updating the balance).
- Check whether company name and its pan number and other details are provided in case of salary account.
- Check whether primary account number is provided in case of secondary account.
- Check whether company details are provided in cases of company's current account.
- Check whether proofs for joint account are provided in case of joint account.
- Check whether you are able deposit an account in the name of either of the person in a joint account.
- Check whether you are able withdraws an account in the name of either of the person in a joint account.
- Check whether you are able to maintain zero balance in salary account.
- Check whether you are not able to maintain zero balance (or mini balance) in non-salary account.

Viva Questions

1. Write about design test cases?
2. What are integration bugs?
3. Define system bugs?
4. What are design specifications?
5. What is path testing?

Experiment No-5 **Date: / / 20**

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

Procedure:

The Library Management System is an online application for assisting a librarian imagining book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification .This test report is the result for testing in the LMS. It mainly focuses on two problems

1. What we will test

2. How we will test.

1. GUI test
Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties
2. Database test

Pass criteria: Results of all basic and advanced operations are normal (refer to section 4)
3. Basic function test
Add a student
Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and Phone number.

The retrieved customer information by viewing customer detail should contain the four attributes.
4. Update/delete student
The record would be selected using the student ID

Updates can be made on full. Items only: Name, Address, Phone number The record can be deleted if there are no books issued by user. The updated values would be reflected if the same customer's ID/SSN is called for.

5. Check-in book
Librarians can check in a book using its call number
The check-in can be initiated from a previous search operation where user has selected a set of books.

The return date would automatically reflect the current system date.

Any late fees would be computed as difference between due date and return date at rate of 10 cents a day.

BACKGROUND

The Library Management System is an online application for assisting a librarian in managing a book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification.

INTRODUCTION

This test report is the result for testing in the LMS. It mainly focuses on two problems: what we will test and how we will test.

Result

GUI test

Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties

Result: pass

Database test

Pass criteria: Results of all basic and advanced operations are normal

Result: pass

Basic function test

Add a student

Pass criteria:

- Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and Phone number.

  Result: pass

- The retrieved customer information by viewing customer detail should contain the four attributes.

  Result: pass

Update/delete student

Pass criteria:

- The record would be selected using the student ID

  Result: pass

- Updates can be made on full. Items only: Name, Address, Phone number

  Result: pass

- The record can be deleted if there are no books issued by user.

Result: Partially pass. When no books issued by user, he can be deleted. But when there are books

Issued by this user, he was also deleted. It is wrong.

- The updated values would be reflected if the same customer's ID/SSN is called for.

  Result: pass

- If customer were deleted, it would not appear in further search queries.

  Result: pass

Add a book

Pass criteria:

- Each book shall have following attributes: Call Number, ISBN, Title, Author name.

  Result: pass

- The retrieved book information should contain the four attributes.

  Result: pass

Update/delete book

Pass criteria:

- The book item can be retrieved using the call number

  Result: did not pass. Can not retrieve using the call number

- The data items which can be updated are: ISBN, Title, Author name

  Result: pass

- The book can be deleted only if no user has issued it.

  Result: partially pass. When no user has issued it, pass. When there are user having issued it,

  did not pass

- The updated values would be reflected if the same call number is called for

  Result: pass

- If book were deleted, it would not appear in further search queries.

  Result: pass

Search for book

Pass criteria:

- The product shall let Librarian query books' detail information by their ISBN number or Author or Title.

  Result: pass

- The search results would produce a list of books, which match the search parameters with following Details: Call number, ISBN number, Title, Author

Result: pass

- The display would also provide the number of copies which is available for issue

  Result: pass

- The display shall provide a means to select one or more rows to a user-list

  Result: pass

- A detailed view of each book should provide information about check-in/check out status, with the borrower's information.

  Result: pass

- The search display will be restricted to 20 results per page and there would be means to navigate from sets of search results.

  Result: pass

- The user can perform multiple searches before finally selecting a set of books for check in or checkout. These should be stored across searches.

  Result: pass

- A book may have more than one copy. But every copy with the same ISBN number should have same detail information.

  Result: pass

- The borrower's list should agree with the data in students' account

  Result: pass

Check-in book

Pass criteria:

- Librarians can check in a book using its call number

  Result: pass

- The check-in can be initiated from a previous search operation where user has selected a set of books.

  Result: pass

- The return date would automatically reflect the current system date.

  Result: did not pass.

- Any late fees would be computed as difference between due date and return date at rate of 10 cents a day.

  Result: did not pass

- A book, which has been checked in once, should not be checked in again

  Result: pass

Check-out book

Pass criteria:

- Librarians can check out a book using its call number

  Result: pass

- The checkout can be initiated from a previous search operation where user has selected a set of books.

  Result: pass

- The student ID who is issuing the book would be entered

  Result: pass

- The issue date would automatically reflect the current system date.

  Result: did not pass

- The due date would automatically be stamped as 5 days from current date.

  Result: did not pass

- A book, which has been checked out once, should not be checked out again

  Result: pass

- A student who has books due should not be allowed to check out any books

  Result: did not pass

- The max. No of books that can be issued to a customer would be 10. The system should not allow checkout of books beyond this limit.

  Result: pass

View book detail

Pass criteria:

- This view would display details about a selected book from search operation

  Result: pass

- The details to be displayed are: Call number, IBN, Title, Author, Issue status (In library or checked out), If book is checked out it would display, User ID & Name, Checkout date, Due date

  Result: for checkout date and due date, did not pass

- Books checked in should not display user summary

  Result: pass

- Books checked out should display correct user details.

  Result: pass

View student detail

Pass criteria:

- Librarians can select a user record for detailed view
  Result: pass
- The detail view should show:
  a. User name, ID, Address & Phone number

  Result: pass

  b. The books issued by user with issue date, due date, call number, title

  Result: did not pass

  c. Late fees & Fines summary and total

  Result: did not pass
- The display should match existing user profile
  Result: pass
- The books checked out should have their statuses marked
  Result: pass
- The book search query should show the user id correctly.
  Result: pass

Network test

Pass criteria: Results of operations (ping, ftp and ODBC connectivity check) are normal

Result: did not test this item, because no enough machines and no available environment.

Viva Questions

1. What are domain bugs?
2. What is meant by predicate coverage?
3. Define path sensitization?
4. What C1, C2 Coverage?
5. Define Link marks and Link counters

Experiment No-6 **Date: / / 20**

Aim : Study of Any Testing Tool(Win Runner)

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).

Win Runner: Win Runner is a program that is responsible for the automated testing of software. Win Runner is a Mercury Interactive enterprise functional testing tool for Microsoft windows applications. Importance of Automated Testing: Reduced testing time Consistent test procedures – ensure process repeatability and resource independence. Eliminates errors of manual testing. Reduces QA cost – Upfront cost of automated testing is easily recovered over the life-time of the product .Improved testing productivity – test suites can be run earlier and more often Proof of adequate testing .For doing Tedious work – test team members can focus on quality areas.

Win Runner Uses:

1. With Win Runner sophisticated automated tests can be created and run on an application. A series of wizards will be provided to the user, and these wizards can create tests in an automated manner.

2. Another impressive aspect of Win Runner is the ability to record various interactions, and transform them into scripts. Win Runner is designed for testing graphical user interfaces. When the user make an interaction with the GUI, this interaction can be recorded. Re-cording the interactions allows determining various bugs that need to be fixed. When the test is completed, Win Runner 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.

3. Win Runner 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.

4. 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.

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

Win Runner 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, Win Runner 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. Win Runner emulates a user by moving the mouse pointer over your application, selecting objects, and entering keyboard input. Win Runner 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, Win Runner retraces the mouse tracks. Use Analog mode when exact mouse coordinates are important to your test, such as when testing a drawing application.

The Win Runner Testing Process Testing with Win Runner involves six main stages:

1. Create the GUI Map The first stage is to create the GUI map so Win Runner can recognize the GUI objects in the application being tested. Use the Rapid Test 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, Win Runner 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 Win Runner 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 Win Runner 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, Win Runner captures them as actual results.

5. View Results Following each test run, Win Runner 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 and 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 Win Runner report view or in the Unified report view. The Win Runner 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 Quick Test 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.

To start Win Runner: **Choose Programs>Win Runner>Win Runner** on the Start menu. The first time you start Win Runner, the Welcome to Win Runner window and the What_s New in Win Runner help open. From the Welcome window you can create a new test, open an existing test, or view an overview of Win Runner in your default browser. If you do not want this window to appear the next time you start Win Runner, clear the **Show on Startup** check box. To show the **Welcome** to **Win Runner window** upon startup from within Win Runner, **choose Settings > General Options**, click the Environment tab, and select the **Show Welcome screen check box**.

The Main Win Runner Window

The main Win Runner window contains the following key elements:

1. Win Runner title bar
2. Menu bar, with drop-down menus of Win Runner commands
3. Standard toolbar, with buttons of commands commonly used when running a test
4. User toolbar, with commands commonly used while creating a test
5. Status bar, with information on the current command, the line number of the insertion point and the name of the current results folder
6. 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 Win Runner 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 reopen Win Runner, 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 soft keys. Sof keys are keyboard shortcuts for carrying out menu commands. You can configure the softkey combinations for your keyboard using the Softkey Configuration utility in your Win Runner program group. For more information, see the Win Runner at a Glance chapter in your Win Runner User‘s Guide. Now that you are familiar with the main Win Runner window, take a few minutes to explore these window components before proceeding to the next lesson.

The Test Window

You create and run Win Runner tests in the test window. It contains the following key elements:

1. Test window title bar, with the name of the open test
2. Test script, with statements generated by recording and/or programming in TSL, Mercury Interactive_s Test Script Language.
3. 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
4. Insertion point, which indicates where you can insert or edit text.

Create a script by recording in Context Sensitive mode that tests the process

1. Start Win Runner.

If Win Runner is not already open, choose Programs > Win Runner > Win Runner on the Start menu.

2. Open a new test.

If the Welcome window is open, click the New Test button. Otherwise, choose File > New. A new test window opens in Win Runner.

3. Start the Flight Reservation application and log in.

Choose Programs > Win Runner > Sample Applications > Flight 1A on the Start menu. In the Login window, type your name and the password mercury, and click OK. The name you type must be at least four characters long. Position the Flight Reservation application and Win Runner so that they are both clearly visible on your desktop.

4. Start recording in Context Sensitive mode.

In Win Runner, choose Create > Record—Context Sensitive or click the Record button on the toolbar. From this point on, Win Runner records all mouse clicks and keyboard input. Note that the text, ―Rec‖ appears in blue above the recording button. This indicates that you are recording in Context Sensitive mode. The status bar also informs you of your current recording mode.

5. Open order #3.

In the Flight Reservation application, choose File > Open Order. In the Open Order dialog box, select the Order No. check box. Type 3 in the adjacent box, and click OK. Watch how Win Runner generates a test script in the test window as you work.

6. Stop recording.

In Win Runner, choose Create > Stop Recording or click the Stop button on the toolbar.

7. Save the test.

Choose File > Save or click the Save button on the toolbar. Save the test as lesson3 in a convenient location on your hard drive. Click Save to close the Save Test dialog box. Note that Win Runner saves the lesson3 test in the file system as a folder, and not as an individual file. This folder contains the test script and the results that are generated when you run the test. Output: Win Runner Test Results window is open and displays the test results. Conclusion: Recording in Context Sensitive mode is cleared and test results are also seen.

Viva Questions

1. What is Manual testing?
2. Give any three automated testing tools?
3. Write the examples of automation testing?
4. Tell the features about Win Runner?
5. How do you run a test in Win Runner

Experiment No-7 **Date: / / 20**

Aim:

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.

c. 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.

Synchronizing test When you run tests, your application may not always respond to input with the same speed. For example, it might take a few seconds:

1. To retrieve information from a database
2. For a window to pop up
3. For a progress bar to reach 100%
4. For a status message to appear

Win Runner waits a set time interval for an application to respond to input. The default wait interval is up to 10 seconds. If the application responds slowly during a test run, Win Runner‘s default wait time may not be sufficient, and the test run may unexpectedly fail. If you discover a synchronization problem between the test and your application, you can either: • Increase the default time that Win Runner waits. To do so, you change the value of the Timeout for Checkpoints and CS Statements option in the Run tab of the General Options dialog box(Settings > General Options). This method affects all your tests and slows down many other Context Sensitive operations. Insert a synchronization point into the test script at the exact point where the problem occurs. A synchronization point tells Win Runner to pause the test run in order to wait for a specified response in the application. This is the recommended method for synchronizing a test with your application. In the following exercises you will:

1. Create a test that opens a new order in the Flight Reservation application and inserts the order into the database
2. Change the synchronization settings
3. Identify a synchronization problem
4. Synchronize the test
5. Run the synchronized test

Input: Creating a test

1. Start Win Runner and open a new test. If Win Runner is not already open, choose **Programs > Win Runner > Win Runner** on the Start menu. If the Welcome window is open, click the New Test button. Otherwise**, choose File > New**. A new test window opens.
2. Start the Flight Reservation application and log in. **Choose Programs > Win Runner > Sample Applications > Flight 1A** on the Start menu. In the Login window, type your name and the password mercury, and click OK. Reposition the Flight Reservation application and Win Runner so that they are both clearly visible on your desktop.
3. Start recording in Context Sensitive mode**. Choose Create > Record Context Sensitive** or click the Record button on the tool bar. Win Runner will start recording the test.
4. Create a new order. **Choose File > New Order** in the Flight Reservation application.
5. Fill in flight and passenger information.
6. Insert the order into the database. Click the Insert Order button. When the insertion is complete, the ―Insert Done‖ message appears in the status bar.
7. Delete the order. Click the Delete Order button and click Yes in the message window to confirm the deletion.
8. Stop recording. **Choose Create > Stop Recording** or click the Stop button
9. Save the test. **Choose File > Save**. Save the test as lesson4 in a convenient location on your hard drive. Click Save to close the Save Test dialog box.

Checking GUI Objects

Input: Adding GUI Checkpoints to a Test Script In this exercise you will check that objects in the Flight Reservation Open Order dialog box function properly when you open an existing order.

Start Win Runner and open a new test.

If Win Runner is not already open, **choose Programs > Win Runner** > **Win Runner** on the Start menu. If the Welcome window is open, click the New Test button. Otherwise, choose File > New. A new test window opens.

Start the Flight Reservation application and log in.

Choose Programs > Win Runner > Sample Applications > Flight 1A on the Start menu. In the Login window, type your name and the password mercury, and click OK. Reposition the Flight Reservation application and Win Runner so that they are both clearly visible on your desktop

Start recording in Context Sensitive mode. Choose Create > Record Context Sensitive or click the Record button on the toolbar.

Open the Open Order dialog box. Choose File > Open Order in the Flight Reservation application.

Create a GUI checkpoint for the Order No.

check box. Choose Create > GUI Checkpoint > For Object/Window, or click the GUI Checkpoint for Object/Window button on the User toolbar .Use the pointer to double click the Order No. check box. The Check GUI dialog box opens and displays the available checks. Note that this dialog box does not open if you only single clicked the Order No. check box. Accept the default check, ―State.‖ This check captures the current state (off) of the check box and stores it as expected results. Click OK in the Check GUI dialog box to insert the checkpoint into the test script. The checkpoint appears as an obj check gui statement.

Enter "4" as the Order No.

Select the Order No. check box and type in 4 in the Order No. text box.

Create another GUI checkpoint for the Order No. check box. Choose Create > GUI Checkpoint > For Object/Window or click the GUI Checkpoint for Object/Window button on the User toolbar. Use the pointer to single click the Order No. check box. Win Runner immediately inserts a checkpoint into the test script (an obj_check_gui statement) that checks the default check ―State.‖ (Use this shortcut when you want to use only the default check for an object.) This check captures the current state (on) of the check box and stores it as expected results.

Create a GUI checkpoint for the Customer Name check box. Choose Create > GUI Checkpoint > For Object/Window or click the GUI Checkpoint for Object/Window button on the User toolbar. Use the pointer to double click the Customer Name check box. The Check GUI dialog box opens and displays the available checks. Accept the default check ―State‖ and select ―Enabled‖ as an additional check. The State check captures the current state (off) of the check box; the Enabled check captures the current condition (off) of the check box. Click OK in the Check GUI dialog box to insert the checkpoint into the test script. The checkpoint appears as an obj_check_gui statement.

Click OK in the Open Order dialog box to open the order.

Stop recording. Choose Create > Stop Recording or click the Stop button.

Save the test. Choose File > Save or click the Save button. Save the test as lesson5 in a convenient location on your hard drive. Click Save to close the Save Test dialog box.

If you are working in the Global GUI Map File mode, save the new objects to the GUI map. Choose Tools > GUI Map Editor. Choose View > GUI Files. Choose File > Save. Click Yes or OK to add the new object or new window to your GUI map. Choose File > Exit to close the GUI Map Editor.

Bitmap checkpoint compares captured bitmap images pixel by pixel.

I**nput**:

Checking Bitmap Objects Adding Bitmap Checkpoints to a Test Script

In this exercise you will test the Agent Signature box in the Fax Order dialog boxYou will use a bitmap checkpoint to check that you can sign your name in the box Then you will use another bitmap checkpoint to check that the box clears when you click the Clear Signature button.

1. Start Win Runner and open a new test. If Win Runner is not already open, choose Programs > WinRunner > WinRunner on the Start menu. If the Welcome window is open, click the New Test button. Otherwise, choose File > New. A new test window opens.

2. Start the Flight Reservation application and log in. Choose Programs > Win Runner > Sample Applications > Flight 1A on the Start menu. In the Login window, type your name and the password mercury, and click OK. Reposition the Flight Reservation application and Win Runner so that they are both clearly visible on your desktop.
3. Start recording in Context Sensitive mode. Choose Create > Record—Context Sensitive or click the Record button on the toolbar
4. **Open order #6.** In the Flight Reservation application, choose File > Open Order. In the Open Order dialog box, select the Order No. check box and type ―6‖ in the adjacent box. Click OK to open the order.
5. **Open the Fax Order dialog box**. Choose File > Fax Order.
6. **Enter a 10digit fax number in the Fax Number box**. You do not need to type in parentheses or dashes.
7. **Move the Fax Order dialog box**. Position the dialog box so that it least obscures the Flight Reservation window.
8. **Switch to Analog mode**. Press F2 on your keyboard or click the Record button to switch to Analog mode.
9. **Sign your name in the Agent Signature box**.
10. **Switch back to Context Sensitive mode**. Press F2 on your keyboard or click the Record button to switch back to Context Sensitive mode.
11. **Insert a bitmap checkpoint that checks your signature**. Choose Create > Bitmap Checkpoint > For Object/Window or click the Bitmap Checkpoint for Object/Window button on the User toolbar.Use the pointer to click the Agent Signature box. Win Runner captures the bitmap and inserts an obj_check_bitmap statement into the test script.
12. **Click the Clear Signature button**. The signature is cleared from the Agent Signature box.
13. **Insert another bitmap checkpoint that checks the Agent Signature box**

Choose Create > Bitmap Checkpoint > For Object/Window or click the Bitmap Checkpoint for Object/Window button on the User toolbar.Use the pointer to click the Agent Signature box. Win Runner captures a bitmap and inserts an obj_check_bitmap statement into the test script.

14. **Click the Cancel button on the Fax Order dialog box**.
15. **Stop recording**. Choose Create > Stop Recording or click the Stop button.

16. **Save the test**. Choose File > Save or click the Save button. Save the test as lesson6 in a convenient location on your hard drive. Click Save to close the Save Test dialog box.
17. **If you are working in the Global GUI Map File mode, save the new objects to the GUI map.** Choose Tools > GUI Map Editor. Choose View > GUI Files. Choose File >Save. Click Yes or OK to add the new object or new window to your GUI map. Choose File > Exit to close the GUI Map Editor.

Viva Questions

1. Define win runner?

2. Explain uses of win runner?

3. What are different modes of win runner?

4. Explain win runner testing process?

5. How to test a module using win runner?

Experiment No-8 **Date: / / 20**

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

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

Creating data driven test

Converting Your Test to a Data Driven Test

Start by opening the test you already created and using the Data Driver Wizard to parameterize the test.

1. **Create a new test from the lesson7 test**.
   If WinRunner is not already open, choose Programs > WinRunner > WinRunner on the Start menu.If the Welcome window is open, click the Open Test button. Otherwise, choose File > Open and select the test you created previously. The test opens. Choose File > Save As and save the test as lesson8 in a convenient location on your hard drive.
2. **Run the Data Driver Wizard.**
   Choose Tools > Data Driver Wizard. The Data Driver Wizard welcome window opens. Click Next to begin the parameterization process.
3. **Create a data table for the test.**
   In the Use a new or existing Excel table box, type ―lesson8‖. The Data Driver Wizard creates an Excel table with this name and saves it the test folder.
4. **Assign a table variable name.**
   Accept the default table variable name, ―table‖. At the beginning of a data driven test, the Excel data table you wish to use is assigned as the value of the table variable. Throughout the script, only the table variable name is used. This makes it easy for you to assign a different data table to the script at a later time without making changes throughout the script.
5. **Select global parameterization options**. Select Add statements to create a data driven test. This adds TSL statements to the test that define the table variable name, open and close the data table, and run the appropriate script selection in a loop for each row in the data table. Select parameterize the test and choose the Line by line option. When you select Parameterize the test, you instruct Win Runner to find fixed values in recorded statements and selected checkpoints and to replace them with parameters. The Line by line option instructs the wizard to open a screen for each line of the selected test that can be parameterized so that you can choose whether or not to parameterize that line. Click Next.
6. **Select the data to parameterize**. The first line byline screen opens. It refers to the Order Number radio button. Adding Data to the Data Table Now that you have parameterized your test, you are ready to add the data the parameterized test will use.
7. **Open the data table**. Choose Tools > Data Table. The lesson8.xls table opens. Note that there is one column named ―Order_Num‖, and that the first row in the column contains value ―4‖.
   a. **Add data to the table**. In rows 2, 3, 4, and 5 of the Order_Num column, enter the values, ―1‖, ―6‖, and ―10‖ respectively.

b. **Save and close the table**. Click an empty cell and choose File > Save from the data table menu. Then choose File > Close to close the table.
c. Save the test. Choose File > Save or click the Save button. Click Save to close the Save Test dialog box.

Adding GUI Objects to the GUI Map Note:

If you are working in the GUI Map File per Test mode, skip this exercise,since new objects are saved in your test's GUI map automatically when you save your test.If your application contains new objects, you can add them to the GUI map without running the RapidTest Script wizard again. You simply use the Learn button in the GUI Map Editor to learn descriptions of the objects. You can learn the description of a single object or all the objects in a window. In this exercise you will add the objects in the Flight Reservation Login window to the GUI map.

1. **Open the Flight Reservation Login window**. Choose Programs > Win Runner > Sample Applications > Flight 1A on the Start menu.
2. **Open the GUI map**. In Win Runner, choose Tools > GUI Map Editor. The GUI Map Editor opens.
3. **Learn all the objects in the Login window**. Click the Learn button. Use the pointer to click the title bar of the Login window. A message prompts you to learn all the objects in the window. Click Yes. Watch as Win Runner learns a description of each object in the Login window and adds it to the temporary GUI Map.
4. **Save the new objects in the GUI map**. Choose Tools > GUI Map Editor. Choose View > GUI Files. Choose File >Save. Click Yes or OK to add the new object or new window to your GUI map. Choose File > Exit to close the GUI Map Editor. 5 Close the Login window. Click Cancel.

Viva Questions

1. Define context sensitive mode?

2. What is test script?

3. What is data driver wizard?

4. How to create test in win runner?

5. How to debug test in win runner?

Experiment No-9: **Date : / / 20**

Aim:

a) How to create a batch test that automatically runs the tests.
b) How to update the GUI object descriptions which in turn supports test scripts as the application changes.

changes in your application

Input:

Maintaining test script

Editing Object Descriptions in the GUI Map Suppose that in a new version of the Flight Reservation application, the Insert Order button is changed to an Insert button. In order to continue running tests that use the Insert Order button, you must edit the label in the button‘s physical description in the GUI map. You can change the physical description using regular expressions.

Start Win Runner and open a new test. If Win Runner is not already open, choose Programs > Win Runner > Win Runner on the Start menu. If the Welcome window is open, click the New Test button. Otherwise, choose File > New. A new test window opens. If you are working in the GUI Map File per Test mode, open the lesson4 test.

Open the GUI Map Editor.

Choose Tools > GUI Map Editor. The GUI Map Editor opens. Make sure that View > GUI Map is selected. The Windows/Object list displays the current contents of the GUI Map. (If you are working in the GUI Map File per Test Mode, the GUI Map Editor will contain fewer objects than as shown below).The GUI Map Editor displays the object names in a tree. Preceding each name is an icon representing the object‘s type. The objects are grouped according to the window in which they are located. You can double click a window icon to collapse or expand the view of its objects.

Find the Insert Order button in the tree. In the GUI Map Editor, choose View > Collapse Objects Tree to view only the window titles.(If you are working in the GUI Map File per Test Mode, the GUI Map Editor will contain fewer objects than as shown below. Double

click the Flight Reservation window to view its objects. If necessary, scroll down the alphabetical object list until you locate the Insert Order button.

View the Insert Order button's physical description. Click the Insert Order button in the tree. (If you are working in the GUI Map File per Test Mode, the GUI Map Editor will contain fewer objects than as shown below.)The physical description object is displayed in the bottom pane of the GUI Map Editor.

Modify the Insert Order button's physical description. Click the Modify button or double click the Insert Order button. The Modify dialog box opens and displays the button's logical name and physical description. In the Physical Description box, change the label property from Insert Order to Insert. Click OK to apply the change and close the dialog box.

Close the GUI Map Editor. In the GUI Map Editor, choose File > Save to save your changes and then choose File > Exit. If you are working in the GUI Map File per Test Mode, choose File > Exit in the GUI Map Editor and then File > Save in Win Runner.

The next time you run a test that contains the logical name ―Insert Order‖, WinRunner will locate the Insert button in the Flight Reservation window. If you are working in the GUI Map File per Test Mode, go back and perform steps1 through 6 for the lesson9 test. In practice, all maps containing the modified object/window must be changed.

Updating the Map with the Run GUI Wizard Note:

If you are working in the GUI Map File per Test mode, skip this exercise,since new objects are automatically saved in your test's GUI map when you save your test.During a test run, if Win Runner cannot locate an object mentioned in the test script, the Run wizard opens. The Run wizard helps you update the GUI map so that your tests can run smoothly. It prompts you to point to the object in our application, determines why it could not find the object, and then offers a solution. In most cases the Run wizard will automatically modify the object description in the GUI map or add a new object description.

For example, suppose you run a test that clicks the Insert Order button in the Flight Reservation window: button_press ("Insert Order");If the Insert Order button is changed to an Insert button, the Run wizard opens during a test run and describes the problem.You click the hand button in the wizard and click the Insert button in the Flight Reservation

program. The Run wizard then offers a solution: When you click OK, Win Runner automatically modifies the object‘s physical description in the GUI map and then resumes the test run.

1. **Open the GUI map**. a. Choose Tools > GUI Map Editor. Choose View > GUI Files.
2. **Delete the “Fly From” list object from the GUI Map Editor tree**. a. The Fly From object is listed under the Flight Reservation window. Select this object and click the Delete button in the GUI Map Editor.
3. **Open Flight Reservation** 1A. a. Choose Programs > Win Runner > Sample Applications > Flight 1A on the Start menu. In the Login window, type your name and the password mercury, and click OK. Reposition the Flight Reservation application and Win Runner so that they are both clearly visible on your desktop.
4. **In Win Runner, open the lesson4 test and run it**. a. Watch what happens when WinRunner reaches the statement list_select_item ("Fly From:", "Los Angeles")
5. **Follow the Run wizard instructions**. a. The Run wizard asks you to point to the Fly From object and then adds the object description to the GUI map. Win Runner then continues the test run.
6. **Find the object description in the GUI map.** a. When Win Runner completes the test run, return to the GUI Map Editor and look for the Fly From object description. You can see that the Run wizard has added the object to the tree.
7. **Close the GUI Map**. a. In the GUI Map Editor, choose File > Exit.
8. **Close the Flight Reservation application**. a. Choose File > Exit. b. Conclusion: GUI map enables using existing test scripts after the user interface changes in application.

Viva Questions

1. Define Batch Testing?

2. What are the different application interfaces?

3. What is data driver wizard?

4. How do you open a GUI Map in win runner?

5. How to update GUI Objects in win runner?

Experiment No -10 **Date: / /20**

Aim: Study of Testing tool Selenium

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 file system path to each test. Consider the following example

```
<html>
<head>
<title>Test Suite Function Tests – Priority 1</title> </head>
<body>
<table>
<tr><td><b>Suite Of Tests</b></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.

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

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 file system path to each test. An example tells it all.

```
<html>
<head>
<title>Test Suite Function Tests – Priority 1</title> </head>
<body>
<table>
<tr><td><b>Suite Of Tests</b></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

Viva Questions

1. Explain about selenium tool?
2. Compare win runner and selenium tool?
3. What are the components of selenium tool?
4. What are test suits for selenium tool?
5. Define selenium grid.

Mini Project:

1.Study of Bug Tracking Tool -Bugzilla

Bugzilla is the most popular bug tracking system available today. Bugzilla's popularity is due to its highly customizable interface, easy configuration, and a large community of very active users. With Bugzilla, you can begin configuring your products, users, and bugs within minutes of starting up the system. Bugzilla provides you with many features, including:

- LDAP integration
- SMTP and Sendmail support
- Internationalized
- User authentication, group security, and SSL support
- Voting system
- Shortcut flags
- Custom fields
- Keyword tagging
- Saved searches
- Voting module
- Multiple report types

In this tutorial, you'll learn how to:

- Set parameters and default preferences
- Create new users
- Create products and components
- Modifying default field values
- Creating new bugs and modifying existing bugs

Setting Parameters and Default Preferences

When you start using Bugzilla, you'll need to set a small number of parameters and preferences. At a minimum, you should change the following items, to suit your 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 you're using. If you're evaluating a click2try application, select **Test**. If you're using SMTP, set any of the other SMTP options for your environment.
6. Click **Save Changes**.
7. In the left side **Index** list, click **Bug Change Policies**.
8. Select On for **commenton create**, which will force anyone who enters a new bug to enter a comment, to describe the bug.

9. Click **Save Changes**.
10. Click **Default Preferences** at the bottom of the page.
11. Select the display order from the drop-down list next to the **When viewing a bug, show comments in this order** field.
12. Click **Submit Changes**.

Creating a New User

Before you begin entering bugs, make sure you add some new users. You 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**. You'll probably want to enable the following options in the row titled

User is a member of these groups:

o **canconfirm**
o **editbugs**
o **editcomponents**

Adding Products

You'll add a product in Bugzilla for every product you are developing. To start with, when you first login to Bugzilla, you'll find a test product called **TestProduct**. You 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 telling you that you'll need to add at least one component.

To add a component:

1. Click the link **add at least one component** in the message that appears after you create a new product.
2. Enter the **Component** name.
3. Enter a **Description**.
4. Enter a **default assignee**. Use one of the users you'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 your product in the message that reads edit other components of product <product name>.

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 you want to edit. The OS field contains a list of operating
system names. You are going to add browsers to this list. In reality, you 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**.
4. In the **Value** field, enter "IE7."
5. Click **Add**.
6. Click **Add a value** again.
7. In the **Value** field, enter "Firefox 3."
8. Click **Add**.
9. Where it reads **Add other values for the op_sys field**, click **op_sys**. This redisplays the table. You 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 you create a new bug.

To create a new bug:
1. In the top menu, click **New**.
2. If you'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 you entered.
5. Select one of the options.
6. Enter a summary and a description. You can add any other information you like, but it is not required by
the system, although you may determine that your bug reporting policy requires certain information.
7. Click **Commit**. Bugzilla adds your bug report to the database and displays the detail page for that bug.

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 you think might be in the title of the bug.
6. Click **Search**. If any bugs meet the criteria you entered, Bugzilla displays them in a list summary.
7. Click the **ID** number link to view the full bug report.

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. Remember it must be in the format of an email address.
3. Click **Commit**.

2. Study of test management tool - Test Director

The Test Director testing process includes four phases:

- Specifying Requirements
- Planning Tests
- Running Tests
- Tracking Defects

Starting Test Director

You start TestDirector from your Web browser, using the TestDirector URL.

To start TestDirector:

1 Open the TestDirector Options window.

In your Web browser, type your TestDirector URL:
http://<TestDirector server name>/<virtual directory name>/default.htm
The TestDirector Options window opens.

2 Open TestDirector.

Click the **TestDirector** link.
The first time you run TestDirector, the application is downloaded to your computer. Then, each time you open TestDirector, it automatically carries out a version check. If TestDirector detects a newer version, it downloads the latest version to your machine.

The TestDirector Login window opens.

3 Select a domain.

In the **Domain** list, select **DEFAULT**.

4 Select a project.

In the **Project** list, select **TestDirector_Demo.**

5 Log on to the project as a QA tester.

In the **User ID** box, type one of the following user names: **alice_td**, **cecil_td**, or **michael_td**. Skip the **Password** box. A password was not assigned to any of the above user names. Click the **Login** button.
TestDirector opens and displays the module in which you were last working. In the title bar, TestDirector displays the project name and your user name.

To explore the TestDirector window:

1 Explore the TestDirector modules.

➢ Click the **Requirements** tab. The Requirements module enables you to specify your testing requirements. This includes defining what you are testing, defining requirement topics and items, and analyzing the
requirements.

➢ Click the **Test Plan** tab. The Test Plan module enables you to develop a test plan based on your testing requirements. This includes defining goals and strategies, dividing your plan into categories, developing tests, automating tests where beneficial, and analyzing the plan.

➢ Click the **Test Lab** tab**.** The Test Lab module enables you to run tests on your application and analyze the results.

➢ Click the **Defects** tab. The Defects module enables you to add defects, determine repair priorities, repair open defects, and analyze the data.

Specifying Testing Requirements

- Defining Requirements
- Viewing Requirements
- Modifying Requirements
- Converting Requirements

Planning Tests

- Developing a Test Plan Tree
- Designing Test Steps
- Copying Test Steps
- Calling Tests with Parameters
- Creating and Viewing Requirements Coverage
- Generating Automated Test Scripts

Running Tests

- Defining Test Sets
- Adding Tests to a Test Set
- Scheduling Test Runs
- Running Tests Manually
- Running Tests Automatically

Adding and Tracking Defects

- Adding New Defects
- Matching Defects
- Updating Defects
- Mailing Defects
- Associating Defects with Tests
- Creating Favorite Views