G. H. Raisoni College of Engineering and Management,
Wagholi, Pune

Department of Computer Engineering

D-19

Lab Manual (2022-23)

Pattern-2020

Class: TY Computer   Term: VI

Software Testing (UCOP306)

Faculty Name: Dr. Vidya Dhamdhere

UCOP306: Software Testing Lab		
Teaching Scheme:	**Credit**	**Examination Scheme**
Practical: 0Hrs./Week	**1**	**INT :25 Marks**

Course Outcomes : On completion of the course, student will be able to–

dentify bugs to create defect report of a given application.

CO2: Summarize test cases for different types and levels of testing.

CO3: Illustrate test plan and Develop and validate a test plan for an application.

CO4: Analyze software using automated testing tools

Sr. No	**List of Laboratory Assignments(*Any 8)**
1	Identify system specification and design test cases for simple calculator application.
2	Design Test cases for Purchase Order Management.
3	Design test cases for Inventory Management.
4	Design test cases for Railway Reservation Form
5	Identify system specification and design test cases for e-commerce(Flipcart, Amazon) login form
6	Write a program and design test cases for the following control and decision making statements For…..Loop Switch….Case Do………While If………Else
7	Prepare test plan for an identified mobile/notepad application.
8	Prepare defect report after executing test cases for library management system/amount withdrawal from ATM machine/any login form.
9	Design and run test cases for Wordpad/MS Word application using automated tool.
10	Open Source Practical

List of Experiments

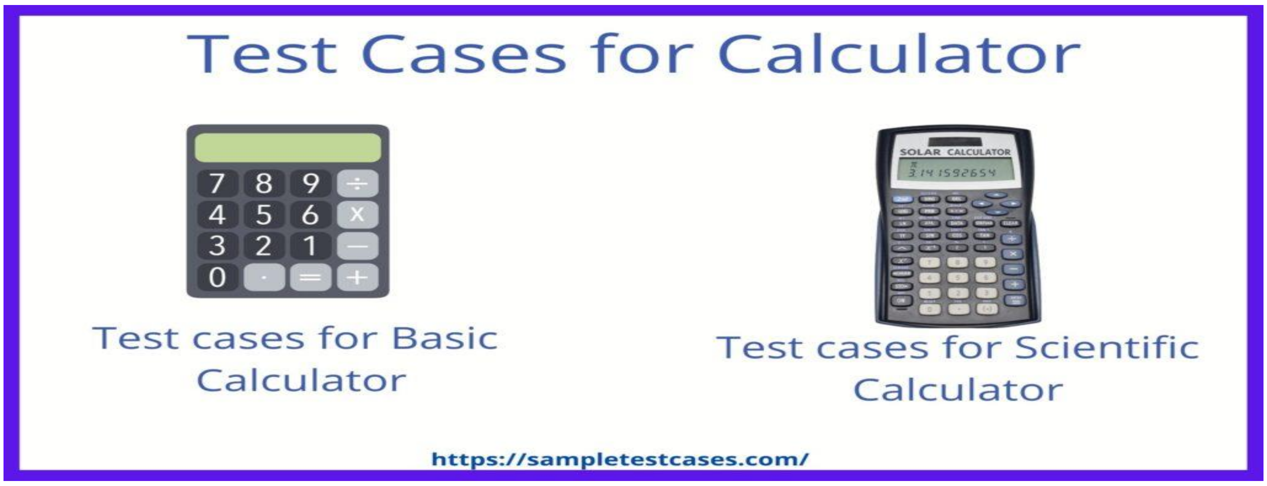
Sr. No.	List of Laboratory Assignments	CO Mapping	Software Required
1	Identify system specification and design test cases for simple calculator application.	CO2	Word/ Excel template
2	Design Test cases for Purchase Order Management.	CO2	
3	Design test cases for Inventory Management.	CO2	
4	Design test cases for Railway Reservation Form	CO2	
5	Identify system specification and design test cases for e-commerce(Flipcart, Amazon) login form	CO1	
6	Write a program and design test cases for the following control and decision making statements For…..Loop Switch….Case Do………While If………Else	CO2	
7	Prepare test plan for an identified mobile/notepad application.	CO3	
8	Prepare defect report after executing test cases for library management system/amount withdrawal from ATM machine/any login form.	CO2	
9	Design and run test cases for Wordpad/MS Word application using automated tool.	CO2,CO4	
	Content beyond syllabus		
10	Automate the test cases using Selenium	CO4	Java/ Python, Selenium

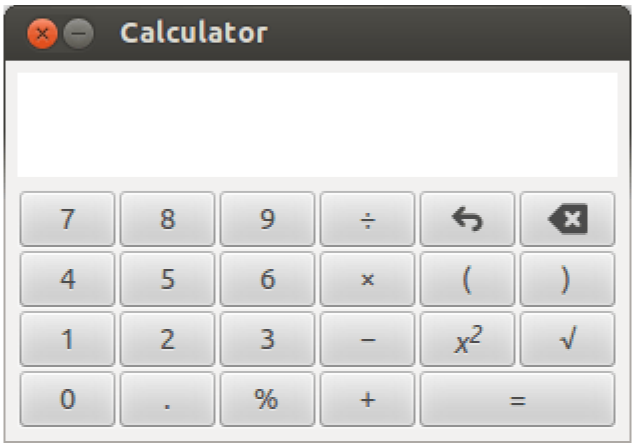
Experiment No.1 :

Title : Identify system specification and design test cases for simple calculator application.

Theory : Test cases for Calculator

Test cases for calculator : In this article, we will discuss test cases for two types of calculator one of simple calculator and second scientific calculator. We include UI test cases, Functional test cases and Negative test cases for the calculators. We can apply functional test scenarios for calculator application.


Test Cases for Calculator
Test cases for Basic Calculator
Test cases for Scientific Calculator
https://sampletestcases.com/



Basic Calculator Test Cases

Basic Operational Tests

Write the test cases based on the following functions and scenarios.

- Check the calculator if it starts by on button. If it is software based calculator then check if it starts via specific means like from searching for calculator in search bar and then executing application. Or by accessing menu item in the Windows.
- Check if the calculator window maximizes to certain window size.

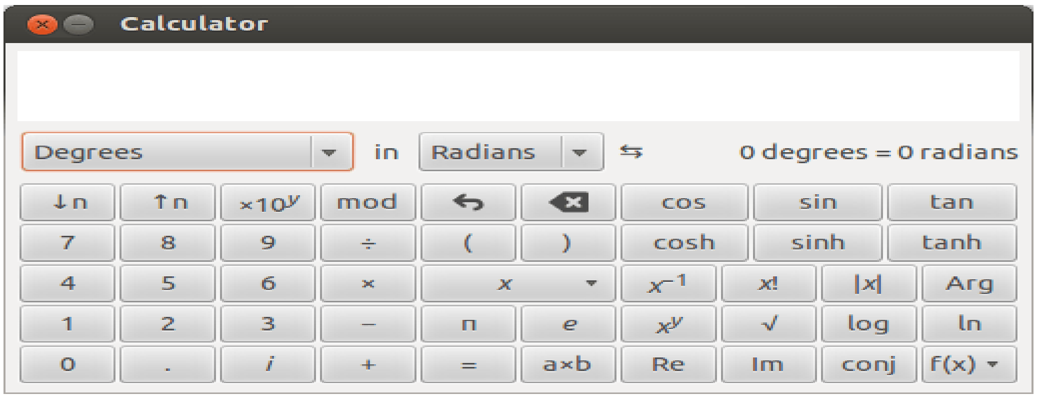
- Check the if the calculator closes when the close button is pressed or if the exit menu is clicked from file > exit option.
- Check if the help document is accessed from Help > Documentation.
- Check if the calculator allows copy and paste functionality.
- Check if the calculator has any specific preferences.
- Check if all the numbers are working (0 to 9)
- Check if the arithmetic keys (+, -, *, %, /) are working.
- Check if the clear key is working.
- Check if the brackets keys are working.
- Check if the sum or equal key is working.
- Check if the square and square root key is working.

Functionality Test Cases

- Check the addition of two integer numbers.
- Check the addition of two negative numbers.
- Check the addition of one positive and one negative number.
- Check the subtraction of two integer numbers.
- Check the subtraction of two negative numbers.
- Check the subtraction of one negative and one positive number.
- Check the multiplication of two integer numbers.
- Check the multiplication of two negative numbers.
- Check the multiplication of one negative and one positive number.
- Check the division of two integer numbers.
- Check the division of two negative numbers.
- Check the division of one positive number and one integer number.
- Check the division of a number by zero.
- Check the division of a number by negative number.
- Check the division of zero by any number.
- Check if the functionality using BODMAS/BIDMAS works as expected.

Advanced Tests on Scientific Calculator

If your calculator has advanced features as shown in the screenshot.



Advanced Calculator Test Cases

You can add few more tests in the scientific calculator.

- Check if the sin, cos, tan and cos is operational using the keys.
- Check if the x-1, x!,|x|,x^y and f(x) is operational and works as expected.
- Check if the log key is operational and works as expected.
- Check if the natural logarithm key i operational and works as expected.
- Check if the factorial key is working as expected.
- Check if the real and imaginary component keys are working as expected.
- Check if the complex conjugate keys are working as expected.

Conversion Function Tests

Some of the advanced scientific calculator has the converter option. It does the conversion of angle, length, weight, area, volume, duration, currency, temperature. Make sure you write the test cases for the same.

Financial Calculator Tests

The additional keys for the financial calculator will be as shown in the image. Some calculator has the mode for enabling these keys.

Test Case template:
Several standard fields for a sample Test Case template are listed below.
Test case ID: Unique ID is required for each test case. Follow some conventions to indicate the types of the test. **For Example,** 'TC_UI_1' indicating 'user interface test case #1'.
Test priority (Low/Medium/High): This is very useful during test execution. Test priorities for business rules and functional test cases can be medium or higher, whereas minor user interface cases can be of a low priority. Testing priorities should always be set by the reviewer.
Module Name: Mention the name of the main module or the sub-module.
Test Designed By Name of the Tester.
Test Designed Date: Date when it was written.
Test Executed By Name of the Tester who executed this test. To be filled only after test execution.
Test Execution Date: Date when the test was executed.
Test Title/Name: Test case title. **For example,** verify the login page with a valid username and password.
Test Summary/Description: Describe the test objective in brief.
Pre-conditions: Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.
Dependencies: Mention any dependencies on other test cases or test requirements.
Test Steps: List all the test execution steps in detail. Write test steps in the order in which they should be executed. Make sure to provide as many details as you can.
Test Data: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.
Expected Result: What should be the system output after test execution? Describe the expected result in detail including the message/error that should be displayed on the screen.
Post-condition: What should be the state of the system after executing this test case?
Actual result: The actual test result should be filled after test execution. Describe the system behavior after test execution.

Status (Pass/Fail): If the actual result is not as per the expected result, then mark this test as **failed**. Otherwise, update it as **passed**.
Notes/Comments/Questions: If there are any special conditions to support the above fields, which can't be described above or if there are any questions related to expected or actual results then mention them here.
Add the following fields if necessary:

Defect ID/Link: If the test status **fails**, then include the link to the defect log or mention the defect number.
Test Type/Keywords: This field can be used to classify tests based on test types. **For Example,** functional, usability, business rules, etc.
Requirements: Requirements for which this test case is being written for. Preferably the exact section number in the requirement doc.
Attachments/References: This field is useful for complex test scenarios in order to explain the test steps or expected results using a Visio diagram as a reference. Provide a link or location to the actual path of the diagram or document.
Automation? (Yes/No): Whether this test case is automated or not. It is useful to track automation status when test cases are automated.

Project Name:	
Test Case Template	
Test Case ID: Fun_10	Test Designed by: <Name>
Test Priority (Low/Medium/High): Med	Test Designed date: <Date>
Module Name: Google login screen	Test Executed by: <Name>
Test Title: Verify login with valid username and password	Test Execution date: <Date>
Description: Test the Google login page	
Pre-conditions: User has valid username and password	
Dependencies:	

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= example@gmail.com	User should be able to login	User is navigated to	Pass	

2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

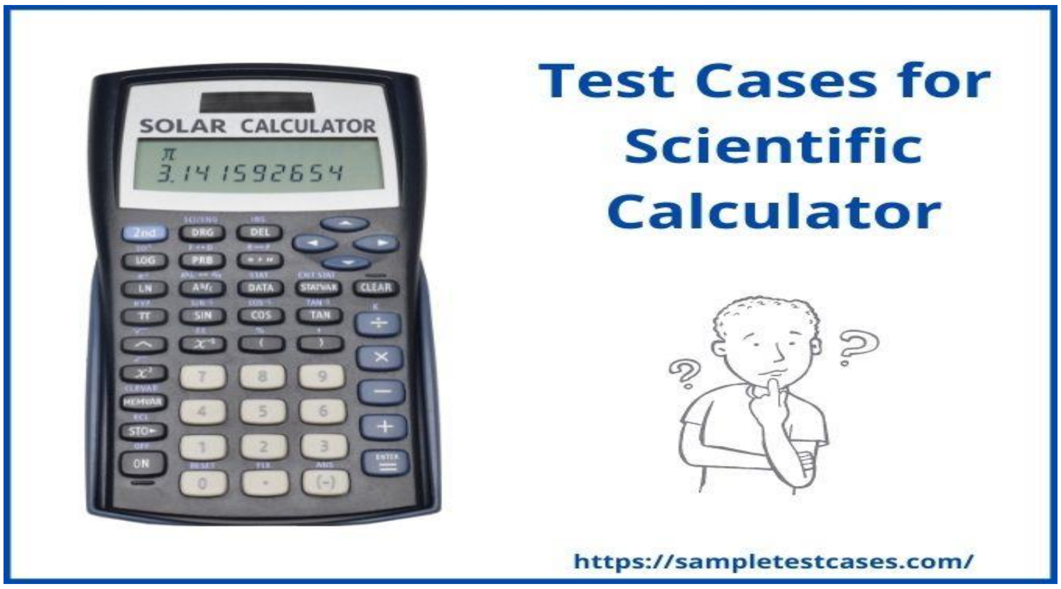
UI Test cases for simple calculator :

Sr.No	**Testcase_ID**	**Test Cases**
1	UI__01	Check that brand name of the calculator
2	UI__02	Check that the brand logo is displayed on the calculator or not.
3	UI__03	Check that color of the calculator
4	UI__04	Check that shape of the calculator
5	UI__05	Check that height of the calculator
6	UI__06	Check that length of the calculator
7	UI__07	Check that weight of the calculator
8	UI__08	Check that material of the calculator
9	UI__09	Check that screen size of the calculator
10	UI__10	Check that buttons are properly displayed or not
11	UI__11	Check that numbers are properly displayed on buttons or not.
12	UI__12	Check that arithmetic signs are properly displayed or not.

13	UI__13	Check that ON and OFF buttons are properly displayed or not.
14	UI__14	Check the size of each of the calculators.
15	UI__15	Check the color of each button of the calculator.
16	UI__16	Check that it has a flip cover or not
17	UI__17	Check the price of the calculator.
18	UI__18	Check that it is operated by battery or solar

Functional Test cases for simple calculator :

2. Test Cases for Scientific Calculator :


SOLAR CALCULATOR
Test Cases for Scientific Calculator
https://sampletestcases.com/

Test cases for scientific calculator

UI Test cases for Scientific calculator :

Sr.No	**Testcase_ID**	**Test Cases**
1	UI_Sci_01	Check that brand name of the calculator
2	UI_Sci_02	Check that the brand logo is displayed on the calculator or not.

3	UI_Sci_03	Check that shape of the calculator
4	UI_Sci_04	Check that screen size of the calculator
5	UI_Sci_05	Check the price of the calculator.
6	UI_Sci_06	Check that it is operated by battery or solar
7	UI_Sci_07	Check the dimension of the calculator.
8	UI_Sci_08	Check the material of the calculator.
9	UI_Sci_09	Check that layout of the calculator
10	UI_Sci_10	Check whether the ON/C and OFF key is displayed properly or not.
11	UI_Sci_11	Check that the MODE key is displayed properly or not.
12	UI_Sci_12	Check the arrows keys are properly displayed or not
13	UI_Sci_13	Check the data entry keys such as 0 to 9 numbers, decimal point, and notation keys are displayed with a proper indication or not.
14	UI_Sci_14	Check that the LCD screen size of the calculator
15	UI_Sci_15	Check that numbers are displayed as expected or not

Functional Test cases for Scientific calculator :

Sr.No	**Testcase_ID**	**Test Cases**
1	Fun_Sci_01	Verify that the initially displayed mode of the calculator

2	Fun_Sci_02	Verify that basic operation such as addition, subtraction, multiplication, and division working as expected or not
3	Fun_Sci_03	Verify that if any user performs a calculation and then press ENTER then the result should be displayed.
4	Fun_Sci_04	Verify that the "=" key is finding the result the same way as ENTER key
5	Fun_Sci_05	Verify that result of the common logarithm calculations
6	Fun_Sci_06	Verify that result of the powers based on the constant e calculation
7	Fun_Sci_07	Verify that result of the factorial calculation
8	Fun_Sci_08	Verify that result of the fractional calculation is expected or not
9	Fun_Sci_09	Verify that the calculation of the trigonometric function as expected or not

Negative Test Cases for Calculator :

Sr.No	**Testcase_ID**	**Test Cases**
1	Negative_TC_01	Check that it is waterproof or not
2	Negative_TC_02	Check that digits are properly displayed or not
3	Negative_TC_03	Verify the result is displayed on the LCD screen or not if the user applies for maximum numbers.

Conclusion: This concludes our post on the test cases for the calculator functionality. I hope these sample test cases will help you in your interviews. In addition, these should help you in writing test cases of similar forms. Please let us know in the comments, how we did. Also, let us know in case we have missed out on anything. As you can see a calculator is a valuable tool for everyone. I hope the above-mentioned test cases will help you to test the calculator object. If you are looking for more examples of test cases then please visit below links.

Experiment No.2

Title: Design Test cases for Purchase Order Management.

Theory:

Testing a Complete Purchase Cycle according to an Example

To familiarize yourself with the system workflow, you will test a purchase-sales workflow in two phases.

The first consists of a product purchase, which requires the following operations:

1. Place a purchase order with Plumbing Component Suppliers for 10 Titanium Alloy Radiators at a unit price of 56.00.
2. Receive these products at your Goods In.
3. Generate a purchase invoice.
4. Pay your supplier.

Then, you will sell some of these products, using this sequence:

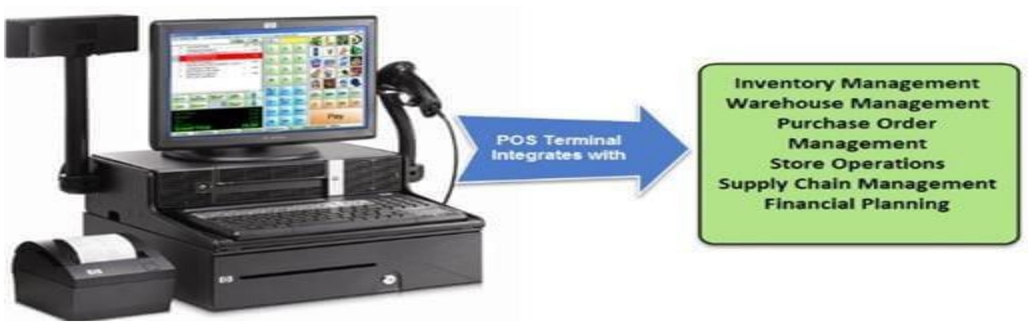
1. Receive a sales order for 6 Titanium Alloy Radiators from Smith and Sons, sold at a unit price of 130.00.
2. Dispatch the products.
3. Invoice the customer.
4. Receive the payment.

Testing Retail Point Of Sale(POS) Systems: Example Test Cases

What is POS Testing?

POS Testing is defined as Testing of a Point of Sale Application. A POS or Point Of Sale software is a vital solution for retail businesses to carry out retail transactions effortlessly from anywhere. You must have seen Point of Sale terminal while checking out at your favorite Mall.

The system is more complex than you think and is tightly integrated with other software systems like Warehouse, Inventory, purchase order, supply chain, marketing, merchandise planning etc. POS Domain Knowledge is important for testing.



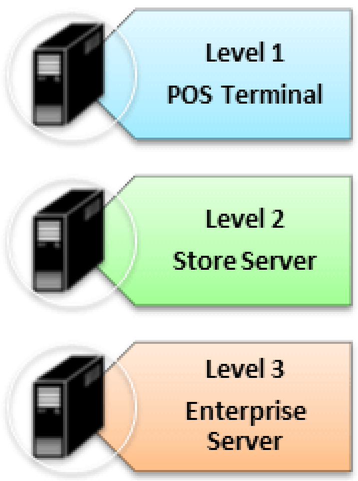
you will learn-

- Test Architecture for POS Application

- Types of Testing for POS system
- Sample Test Cases for POS used in Retail
- Security Testing for Retail POS Systems
- Challenges in POS testing

Test Architecture for POS Application

POS test architecture includes three components for testing – POS terminal, store server, and enterprise server. Basically, it is classified into three levels for testing of POS application.

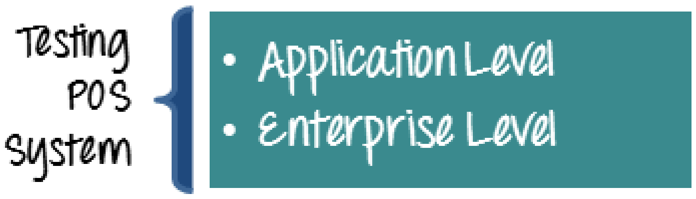


Level 1- (POS Terminal)	Level 2- (Store Server)	Level 3- (Enterprise Server)
• Device and hardware testing (RFID, Scanner, Printer, Barcode reader) • Interoperability Testing • BI and Analytics Testing • Performance Testing	• Security Testing • BI & Analytics Testing • Disaster Recovery Testing • Interface Testing	• Security Testing • BI & Analytics Testing • Disaster Recovery Testing • Interface Testing

Types of Testing for POS system

Testing of POS System can be broken down into two levels

1. Application Level
2. Enterprise Level



Testing Performed At Application Level	Testing Performed At Enterprise Level
• Functionality Testing • Compatibility Testing • Payment Gateway Testing • Report Testing	• Compliance Testing • Performance Testing • Interoperability Testing • Data Migration • Mobility

Sample Test Cases for POS used in Retail

To ensure quality of the POS system, proper POS software testing is mandatory. The POS testing spans many things like

Test Scenario	Test Cases
Cashier activity	• Test the entry of items purchased by a customer is correct • Test discounts are applied correctly • Verify store value cards can be used • Check petty cash management works as expected • Check totals and closings match • Check cash drawer loans are handled properly • Test the POS system is compatible with peripherals like RFID Reader, Bar Code Scanner etc.
Payment Gateway Processing	• Test the validity of CVV number of Credit Card • Test swiping of cards from both sides and chips • Verify that the captured card details are properly encrypted and decrypted
Sales	• Check for a regular sale process • Check sales can be processed with debit/credit cards • Check for loyalty membership purchase • Check for correct prices are displayed for merchandise purchased • Test for “0” or null transaction • Tie UPC or barcodes to vendors • Test for billing details or shipping details in payment manager • Test for reference transaction

	• Test the print format of the receipt generated • Verify that the correct code is generated for approved, hold or declined transactions
Return & Exchange scenarios	• Make sure the in-house inventory is well integrated with other outlets or supply chain • Check for exchange or return of an item with cash • Check whether system responds on exchange or return of an item with a credit card • Check system process the sale with receipt or without a receipt • Verify that system should allow entering bar-code manually incase scanner don't work • Verify system display both the current amount as well as the discount amount on an exchange of item if applicable
Performance	• Check for speed or time taken to receive a response or send a request • Check the transaction based rules are applicable (discounts/tax/ rebates etc.) • Verify that the correct code is generated for approved, hold or declined transactions
Negative Scenarios	• Test system with expired card details • Test with an invalid PIN for credit card • Check the inventory by entering a wrong code for the item • Check how a system responds while entering a wrong invoice number • Test for a negative transaction • Test the response of system while entering an invalid date for promotional offers online items
Managing Promotions and Discounts	• Test system for various discount like a veteran discount, seasonal discount, undergage or overgage discount etc. • Test system for various promotional offers on certain line items • Test alert system that notifies end or beginning of seasonal offers • Test whether receipt print the exact discount or offers that are leveraged • Test system for allocating wrong offers or discount online item • Test the order management process • Verify product data obtained after scanning a barcode is accurate
Tracking customer's data	• Test for system response with incorrect customer data input • Test system for allowing authorized access to customer's confidential data • Test the database for recording customer's buying history like (what they buy, how frequent they buy, etc.)
Security & Regulatory Compliance	• Verifying POS system as per regulatory compliances • Test alert system that notifies security defenders • Make sure you can void a payment before posting • Test user profiles and access levels on the POS Software

	• Test database consistency • Verify specific information about each tender cash, coupon identifier, check number and so on
Report testing	• Testing of a trend analysis report • Test information related to credit card transaction should be reflected in reports • Test for the individual as well as consolidated reports of customers buying history • Test for online report generation

Security Testing for Retail POS Systems

Some recent studies have Point of Sale Systems very high-security vulnerabilities. Following measures will help with security of POS

- Security testing in compliance with PCI standard is very crucial to be addressed as the part of enterprise testing
- Actively manage all software on the network so that only authorized software can only execute and installed
- Conduct regular Penetration Testing to identify attack vectors and vulnerabilities
- Include tests for the presence of unprotected system information and artifacts that would be useful to hackers
- Use vulnerability testing tools
- Create a testbed that imitate a production environment for specific penetration tests and attacks against elements that are not tested in productionChallenges in POS testing

- Multiple Configurations
- Complex interfaces
- Peripheral issues
- Upgrades
- PCI compliance
- Test lab maintenance

Test case ID	Test Scenario	Pre-Conditions	Test Steps	Expected Results	Actual Results	Test Results
TC_1	Verify login and Password.	Browser Launched and Navigate to https://democontosodatadevaos.sandbox.ax. dynamics.com/?cmp=USMF&mi=DefaultDashboard	Enter the valid Username and Password successfully.	Login must be successful.	Login successfully done.	Pass
TC_2	Create new purchase order	Login Successful	1.Click new order in left most part of Dashboard. 2.Enter the Vendor account and site name. 2.Save the details after entering mandatory fields.	Purchase order must be successful.	Purchase order Creation successfully done.	Pass
TC_3	Search Item number	Login Successful	Enter the associated item number for the vendor	Enter the associated item number for the vendor	Item number must be added successfully	Pass
TC_4	Save and confirm Purchase order	Login Successful	Click the save button and confirm the purchase order.	Click the save button and confirm the purchase order.	Purchase order Creation must be successful.	Pass

- **Conclusion:** Retail POS demands a high level of testing keeping in mind that its performance and correct functioning directly affect business revenues.
- To reduce the risk and chances of POS failure during the transaction process, testing under the extreme condition is essential.
- Testing needs to perform at Application as well as Enterprise Level
- Your Testing should cover the following scenarios – Cashier activity, Payment Gateway Processing, Sales, Return & Exchange scenarios, Performance, Negative Scenarios, Managing Promotions and Discounts, Security & Regulatory Compliance.
- Multiple configuration settings, peripheral issues, upgrades are few issues you will need to tide over while testing.

Experiment No.3

Title: Design test cases for Inventory Management.

Theory: The test case generation of Store Management System as example.

Login

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	1
Related UC/FR/NFR	UC/FR/NFR
Date:	02-01-2014
Purpose:	The purpose of this test insure that the user can log in with his id to use this app.
Pre-Req:	The user enters his correct name and password for a successful login page.
Test Data:	sname, sid, spassword
Steps:	The steps are below. 1. visit Login Page 2. enter UserId 3. enter password 4. click login 5. recovery password 6. change password 7. keep login
Status:	Pass

Main form

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	2
Related UC/FR/NFR	UC/FR/NFR
Date:	03-01-2014
Purpose:	The purpose of this test is to make sure that the design of the front page of the app is consistent and readable.
Pre-Req:	The main activity of the app should be up and running.
Test Data:	none (test is just visual test)
Steps:	Following steps will take place in the test 1. Latest orders 2. Top customers 3. Recent products 4. Recent customers

	5. Graph by amount and order 6. Graph by day/month/year
Status:	Pass

Employ Records

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	3
Related UC/FR/NFR	UC/FR/NFR
Date:	03-01-2014
Purpose:	The purpose of this test for registered employ by his name, password, email, and mobile number for use this app and work on the app.
Pre-Req:	Employ sure that login with his name and id to use this app.
Test Data:	none (test is just visual test)
Steps:	Step formatting rules below. 1. enter employ name 2. enter employ password 3. enter employ email 4. enter employ a mobile number
Status:	Pass

Category Record

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	4
Related UC/FR/NFR	UC/FR/NFR
Date:	04-01-2014
Purpose:	The purpose of this testing to insert, update, delete and view category.
Pre-Req:	The user enter category by its correct name.
Test Data:	Txtcatogaryname
Steps:	Step formatting rules below. 1. open the category form 2. insert the category name of items 3. update the category name of items 4. delete the category name of items 5. insert the category name of items 6. view the category names 7. Export the category records to MS Excel.

Status:	Pass

Customer Record

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	5
Related UC/FR/NFR	UC/FR/NFR
Date:	05-01-2014
Purpose:	The purpose of this testing to registered the new customer by his name, address, id number and phone number for sales items.
Pre-Req:	The user can easily search customer by name with all data.
Test Data:	Scustmrname, scsutmradress, scustmrmblnumer, scustmremail
Steps:	Step formatting rules below. 1. automatically generate customer-id 2. enter the customer name 3. enter customer address 4. enter customer id number 5. enter customer city 6. enter customer mobile number 7. enter customer email
Status:	Pass

Items Record And Configuration

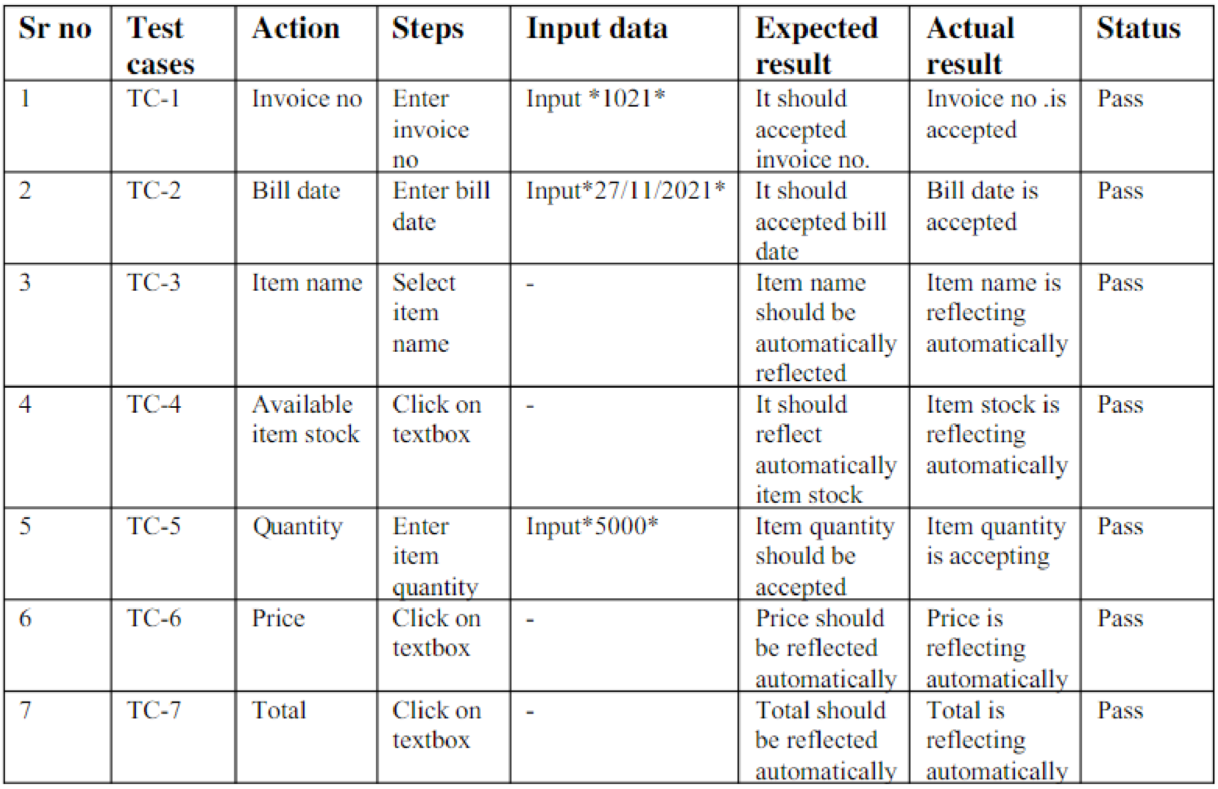
Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	6
Related UC/FR/NFR	UC/FR/NFR
Date:	05-01-2014
Purpose:	The purpose of this testing to enter item code, name and price and also can modify name and price of items.
Pre-Req:	Employ can search items by name and code.
Test Data:	none (test is just visual test)
Steps:	Step formatting rules below. 1. insert items code 2. insert items name with price 3. update items name or price 4. delete items name
Status:	Pass

Sales and Report

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	7
Related UC/FR/NFR	UC/FR/NFR
Date:	05-01-2014
Purpose:	The purpose of this testing to sales items with discount and print invoice of sales.
Pre-Req:	The user can check records for recent or old sales by report or data.
Test Data:	none (test is just visual test)
Steps:	Step formatting rules below. 1. sales items with quantity 2. discount to customer 3. payment due 4. print crystal report of sales 5. print excel report of sales
Status:	Pass

Stock

Test Engineer:	Here, you can mention the Name of the test engineer
Test Case ID:	8
Related UC/FR/NFR	UC/FR/NFR
Date:	05-09-2014
Purpose:	The purpose of this test for check quantities of items in which user can modify items day by day.
Pre-Req:	The user can see the quantities are below in the range or above in the range.
Test Data:	None
Steps:	Step formatting rules below. 1. set the quantities of items 2. update the quantities of items 3. delete the quantities of items 4. if quantities less than 200 than message show with yellow color 5. if quantities equal to zero than message show with red color
Status:	Pass



Sr no	Test cases	Action	Steps	Input data	Expected result	Actual result	Status
1	TC-1	Invoice no	Enter invoice no	Input *1021*	It should accepted invoice no.	Invoice no .is accepted	Pass
2	TC-2	Bill date	Enter bill date	Input*27/11/2021*	It should accepted bill date	Bill date is accepted	Pass
3	TC-3	Item name	Select item name	-	Item name should be automatically reflected	Item name is reflecting automatically	Pass
4	TC-4	Available item stock	Click on textbox	-	It should reflect automatically item stock	Item stock is reflecting automatically	Pass
5	TC-5	Quantity	Enter item quantity	Input*5000*	Item quantity should be accepted	Item quantity is accepting	Pass
6	TC-6	Price	Click on textbox	-	Price should be reflected automatically	Price is reflecting automatically	Pass
7	TC-7	Total	Click on textbox	-	Total should be reflected automatically	Total is reflecting automatically	Pass

Conclusion:

We had study inventory management system and on that basis test cases as design .

Experiment No.4

Title: Design test cases for Railway Reservation Form

Theory: Test Cases For Online Ticket Booking Using IRCTC

- Verify whether the https://www.irctc.co.in/nget/ ticket booking website is loading correctly or not.
- Verify on filling train details like From station, To station, Departure date, and list of available trains are displayed.
- Verify that users can search for trains by name and from to station to check their status and timings.
- Verify that search results have train details, timings, and availability.
- Verify that clicking the search results opens complete details for the train.
- Verify that the user should see real-time train status of availability of seats.
- Verify the user can see available seats on a train.
- Verify that the pricing of different types of seats on a train is displayed to the users.
- Verify User can select a single or more than one seat.
- Verify if the seat is booked, then the user should not be able to purchase the ticket for that seat.
- Verify when the user selects a seat, enters passenger details, and makes the payment. Then, the selected seat should get booked.
- Verify once the user has booked the ticket, then he can download the ticket.
- Verify the status of the seat gets changed to booked after the user has booked the ticket.
- Verify user should receive an SMS or mail after successfully booking of ticket.
- Verify the user can book one ticket per document.
- Verify that his amount should be refunded when the user cancels the ticket.
- Verify when the user cannot cancel the ticket after the train has left the station.
- Verify that when a user tries to cancel the ticket on the day of travel, he should not get the total amount as a refund.
- Verify by trying payment from different types of payment modes.
- Once the user has canceled, the ticket status should change to available.
- Verify error message is displayed when the user enters a unique character from and to the field.
- Verify error message is displayed when the user leaves all the field blank and click on search train.
- Verify error message should be displayed when a user leaves the passenger details form blank and click on submit.
- Verify payment should get rejected when the user enters the wrong credentials during the payment process.
- Verify the user is getting extra benefit if he has purchased a ticket for the AC compartment.
- Verify user should not be able to purchase a ticket if he is not login to the application.
- Verify that if the user tries to book a ticket without logging into the app, he should be redirected to the login page.
- Verify the ticket price mentioned on the user ticket. That amount is only debited from the customer’s account.
- Verify user details are mentioned properly on the ticket the user has downloaded.
- Verify the user can search the train status using the PNR number.
- Verify user’s amount is refunded if his account is debited, but the ticket is not booked due to a network issue.
- Verify the user cannot purchase a ticket for the past date.

- Verify the user is getting a ticket on his what’s app if he has selected the option of sharing a ticket on what’s app.
- Verify after successful booking of ticket user should be able to see his ticket booking history.
- Verify that when a user enters the wrong id proof number, an error message should be displayed.
- Verify if the user has not entered the IRCTC ID, and tries to book a ticket, then an error message should be displayed.
- Verify if the user has not entered the email id during the ticket booking process, then an error message should be displayed.
- Verify that an error message should be displayed if a user has entered a unique character in the name field during the ticket booking process.
- When the user clicks on the edit passenger details button, the Edit Traveller section should open.
- Verify that the user can delete the passenger detail by clicking the delete button.
- Verify when the user has edited his details then it should get updated, old data should not be reflected in the passenger detail section.
- Verify when the user selects some other nationality the passport number field should be visible to the user.
- Verify the user can change his boarding point during the ticket booking process.

Conclusion:

Now We come to the end of this post, but we have covered somany important test cases or test scenarios that are frequently asked in the interviews. But as the applications are huge there may be a chance of missing some of the test scenarios. So if you found any such testing scenarios then you can inform us in the comment section.

Experiment No.5

Title: Identify system specification and design test cases for e-commerce (Flipcart, Amazon) login form

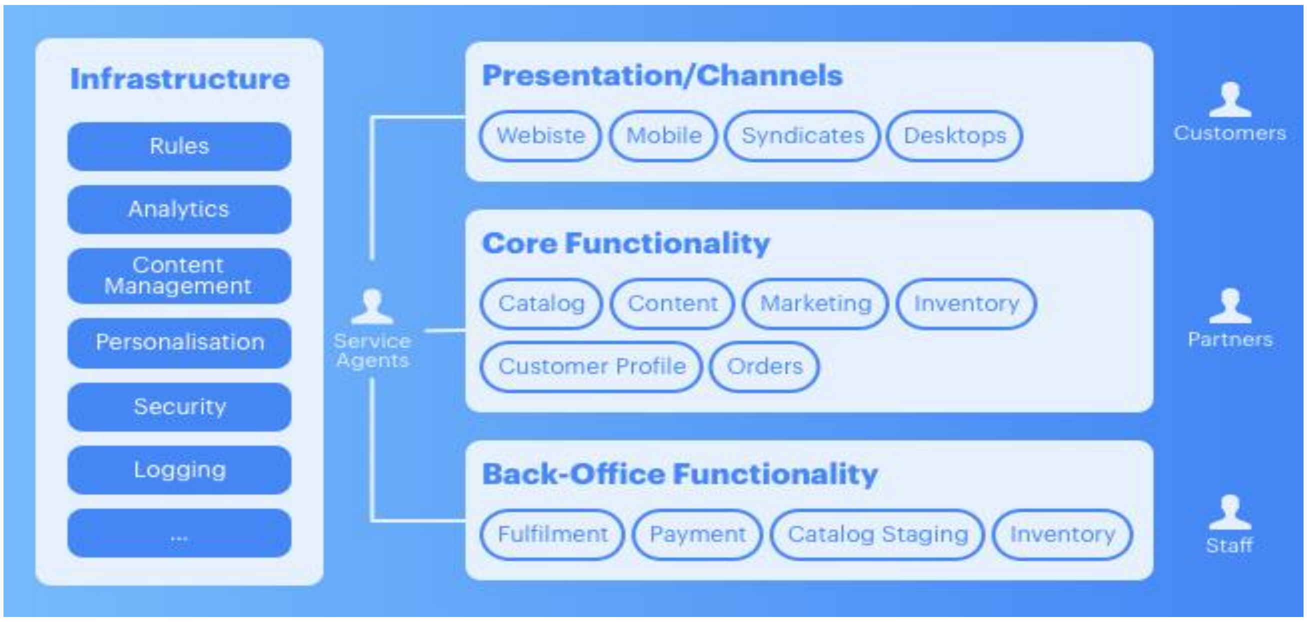
Theory:

Ecommerce application architecture

An eCommerce application has 4 important elements – Main Pages, Product Pages, Product Description Pages, and Shopping Cart. Understand these in detail to test e-commerce websites and applications efficiently.

Main Pages include homepage, privacy page, about us page, careers, etc. Product Page includes different options for the product such as size, color, and other attributes. Sorting and filtering features are considered as part of product type pages along with add-to-cart and wishlist features.Product description page includes title, description, images, add to cart feature and additional info, etc. The shopping cart page should include payment options and removing a product from the cart.Overall, an eCommerce website or an application includes different user roles such as customers, partners, staff, and service agents. Backend infrastructures include rules, analytics, security, logging, and content management.The first layer which is revealed to the users is the presentation layer that includes the UI for mobile, desktops. Responsiveness is kept in mind when designing this presentation layer.

Core Functionality includes Content, Marketing, Inventory Management, Orders, and Catalog. Back-office functionality includes fulfillment, inventory, payment, and Catalog staging. The conceptional view is shown below.



Testing Ecommerce Website Workflow

Testing of the complete workflow is important to test an eCommerce application. Let's look at the components of the workflow such as Login and Signup, Search Functionality, Sorting Feature, Filter Feature, Adding or removing a product to the cart, check out process, Order number generation, Invoice generation, and Payment gateway.

Below are the **test cases for the online shopping system** that will help you in testing your mobile application. These **test scenarios are for an online shopping website** that will cover all pages of your eCommerce application.

General Test Cases for E-commerce Websites & Applications

1. User navigation through all the pages of the application
2. None of the links in the applications should be broken.
3. Company logo, products, prices, and their description should be visible.
4. Products should be listed category-wise on the application.
5. Products should be displayed which match the search criteria.
6. Relevant products should be listed on the top of the search results page.
7. Filtering functionality should work properly i.e., correct products are filtered when the filter is applied.
8. Ensure correct count of products is displayed on search and filter.
9. Sorting should be working correctly on all the pages – the products are sorted based on the sort of option.
10. The product count should remain the same even when sorting is applied.

Product Page Test Cases

1. Users should be able to select the desired attributes of the product-on-product page such as size, color, etc.
2. Adding a product to the cart should be possible
3. Checking whether users can add a product to the wish list.
4. Users should be able to buy the product which is added to the cart once the user is signed in.
5. Customers shouldn't be able to add products to the cart when it is out of inventory.
6. All the products which are added to the cart should be purchasable by the user.
7. Verify error message is displayed on the UI when there is a limit on the products which can be purchased.
8. Error message should be displayed on the UI when shipping is not available to the delivery location.
9. All the payment methods should be displayed and all of the methods should be working correctly.
10. Ensure email gets triggered to the email address or mobile number when a product is bought by the customer.

Payment Gateway Test Cases

1. Verify product price is correct along with shipping charges, VAT. VAT and shipping charges should be correctly applied.
2. Confirm VAT varies based on the number of products in the cart.

3. Verify all the payment methods are correctly working such as net banking, credit/debit card, and PayPal using dummy numbers for testing.
4. Ensure payment is refunded to the customer when a product is cancelled based on payment id.
5. Make sure the emails and invoices sent to the customer after a product is purchased by the user.
6. Verify emails are sent to the customer when the payment is successfully refunded to the user.

Search Functionality Test Cases

1. Correct search results should show up for different types such as product name, brand name, or fuzzy search.
2. Search results should be relevant to the search criteria
3. Different sort options should work correctly after the search is applied.
4. Search results should be displayed as per pagination.
5. Verify search should work correctly based on different functionalities.

Shopping Cart Test Cases

1. User should be able to add a product to the cart.
2. Item count should be incremented when the user adds the same product again.
3. Taxes should be applied according to the delivery location.
4. User should be able to add items to the cart.
5. User should be able to update items in the cart.
6. Checkout should happen successfully for the items added to the cart.
7. Shipping costs for different products added to the cart.
8. Coupons should be applied successfully to the cart.
9. Cart should retain the items even when the app is closed.

Post-Order Test Cases

1. Email and order id should be sent after placement of order.
2. Users should be able to cancel the order.
3. There should be facility for users to track the order.
4. Users should be able to return/replace the product post-delivery.

Test strategy for eCommerce website

While testing an eCommerce application test strategy and test plan should be designed carefully. Let's look at the **test plan for an online shopping cart** which includes different **types of testing needed for an eCommerce website**

1. Browser Compatibility Testing

The application should be tested across main platforms such as Linux, Windows, and Mac.

2. Load and Performance Testing

Ecommerce applications should be tested regressively by applying high load and testers should make sure that the performance of the application is up to the mark. Ecommerce apps are used by a lot of people and there is a high surge in customers on big billion days or any sale days.

Testers should apply the maximum load and then test spikes in the database, the response time of APIs, and the network bandwidth. Testers should know the number of concurrent requests which can be handled by the application. Response time and loading time of the application should be tested in advance. Load balancers should be applied during extensive **load testing.**

3. Functional Testing

All the functionalities of the application should be tested regressively based on different pages in the application. UI of the application is also taken care of in functional testing. Testers should make sure that the application should be responsive.

4. Security Testing

Testers should make sure that the application is secure from all attacks. Al the transactions which happen via your eCommerce application should be secured.

Customer data should not be leaked for data protection policy. There have been instances where data breaches happened in the past which cost businesses a lot of bucks. Hence, security should be the main testing for eCommerce applications.

How to test e-commerce websites manually

The application should be tested manually by covering all pages and all types of testing. UI and functional testing can be done manually by testers by executing all the test cases mentioned above. Effective **test plans for online shopping websites** and creating an effective **test strategy for eCommerce websites** are the heart of testing the application manually. Tracking statuses of test cases and raising bugs early in the SDLC will help in reducing the cost associated with the bugs.

How to test e-commerce websites using **Selenium**

Functional testing of an application can be done using **e-commerce testing tools** such as Appium which is based on top of selenium. Testers can automate almost all functional tests using Appium.

You can do performance testing using the JMeter tool whereas, for security testing ZAP OWASP is a great choice. For automating **Unit testing,** developers can use the core language used for application development.

Conclusion

The above **e-commerce website testing checklist** and **test plan for an eCommerce website** would have given an idea of testing eCommerce applications such as Flipkart.

Do you still have doubts about testing your eCommerce application? Contact us to know more. Get free estimation from us and make your product more robust with minimum bugs.

Testscenario is a software testing team with experienced and efficient software testers. We are equally adept at manual and automated testing. Get in touch with us today, and get the best in the business spot out bugs for you.

Test Cases – Login Page
Following is the possible list of functional and non-functional test cases for a login page:
Functional Test Cases:

Sr. No.	**Functional Test Cases**	**Type- Negative/ Positive Test Case**
1	Verify if a user will be able to login with a valid username and valid password.	Positive
2	Verify if a user cannot login with a valid username and an invalid password.	Negative
3	Verify the login page for both, when the field is blank and Submit button is clicked.	Negative
4	Verify the 'Forgot Password' functionality.	Positive
5	Verify the messages for invalid login.	Positive
6	Verify the 'Remember Me' functionality.	Positive
7	Verify if the data in password field is either visible as asterisk or bullet signs.	Positive
8	Verify if a user is able to login with a new password only after he/she has changed the password.	Positive
9	Verify if the login page allows to log in simultaneously with different credentials in a different browser.	Positive
10	Verify if the 'Enter' key of the keyboard is working correctly on the login page.	Positive
	Other Test Cases	
11	Verify the time taken to log in with a valid username and password.	Performance & Positive Testing
12	Verify if the font, text color, and color coding of the Login page is as per the standard.	UI Testing & Positive Testing

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
13	Verify if there is a 'Cancel' button available to erase the entered text.	Usability Testing
14	Verify the login page and all its controls in different browsers	Browser Compatibility & Positive Testing.

Non-functional Security Test Cases:

Sr. No.	Security test cases	Type- Negative/ Positive Test Case
1	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Negative
2	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Positive
3	Verify the login page by pressing 'Back button' of the browser. It should not allow you to enter into the system once you log out.	Negative
4	Verify the timeout functionality of the login session.	Positive
5	Verify if a user should not be allowed to log in with different credentials from the same browser at the same time.	Negative
6	Verify if a user should be able to login with the same credentials in different browsers at the same time.	Positive
7	Verify the Login page against SQL injection attack.	Negative
8	Verify the implementation of SSL certificate.	Positive

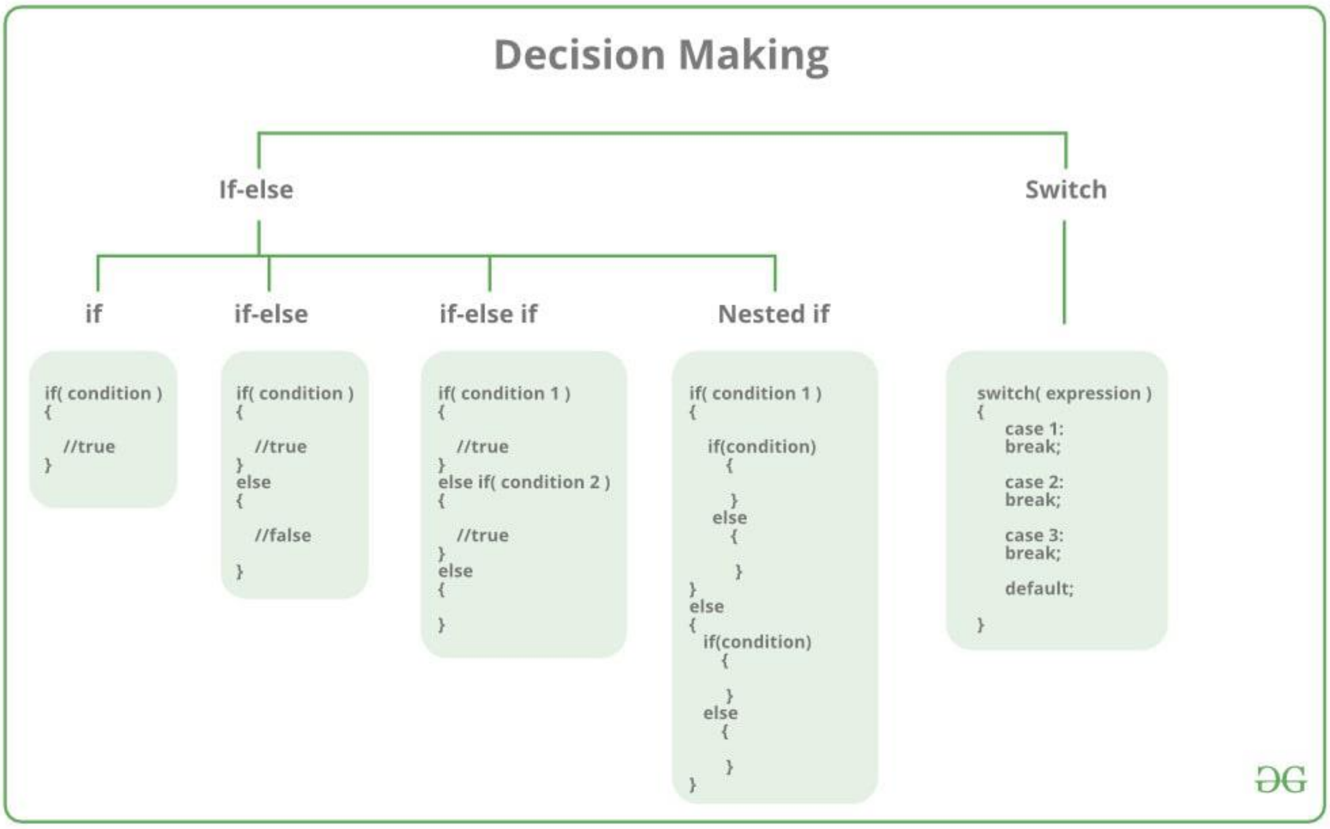
Conclusion: WE had study online web portable working with its test case design.

Experiment No.6

Title: Write a program and design test cases for the following control and decision making statements

1.For…..Loop
2.Switch….Case
3.Do………While
4.If………Else

Theory:



Decision Making
If-else
Switch
if
if-else
if-else if
Nested if
if(condition) { //true }
if(condition) { //true } else { //false }
if(condition 1) { //true } else if(condition 2) { //true } else { }
if(condition 1) { if(condition) { } else { } } else { if(condition) { } else { } }
switch(expression) { case 1: break; case 2: break; case 3: break; default; }

What is an if-else statement?

An if-else statement in C programming is a conditional statement that executes a different set of statements based on the condition that is true or false. The 'if' block will be executed only when the specified condition is true, and if the specified condition is false, then the else block will be executed.

Syntax of if-else statement is given below:

```
if(expression)
{
    // statements;
}
```

```
else
{
   // statements;
}
```

What is a switch statement?

A switch statement is a conditional statement used in C programming to check the value of a variable and compare it with all the cases. If the value is matched with any case, then its corresponding statements will be executed. Each case has some name or number known as the identifier. The value entered by the user will be compared with all the cases until the case is found. If the value entered by the user is not matched with any case, then the default statement will be executed.

Syntax of the switch statement is given below:

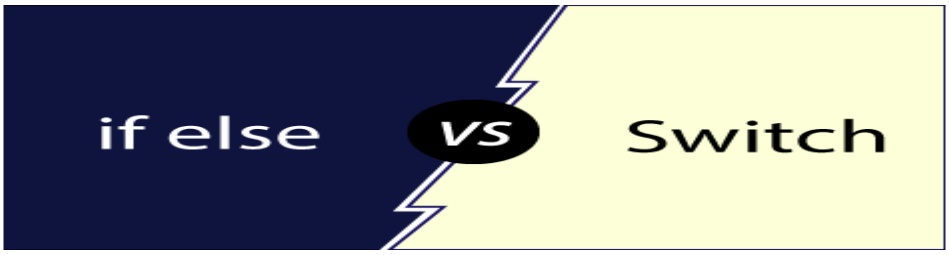
```
switch(expression)
{
  case constant 1:
  // statements;
  break;
  case constant 2:
  // statements;
  break;
  case constant n:
  // statements;
  break;
 default:
// statements;
}
```

Similarity b/w if-else and switch

Both the if-else and switch are the decision-making statements. Here, decision-making statements mean that the output of the expression will decide which statements are to be executed.

Differences b/w if-else and switch statement

The following are the differences between if-else and switch statement are:



- **Definition**

if-else

Based on the result of the expression in the 'if-else' statement, the block of statements will be executed. If the condition is true, then the 'if' block will be executed otherwise 'else' block will execute.

Switch statement

The switch statement contains multiple cases or choices. The user will decide the case, which is to execute.

- **Expression**

If-else

It can contain a single expression or multiple expressions for multiple choices. In this, an expression is evaluated based on the range of values or conditions. It checks both equality and logical expressions.

Switch

It contains only a single expression, and this expression is either a single integer object or a string object. It checks only equality expression.

- **Evaluation**

If-else

An if-else statement can evaluate almost all the types of data such as integer, floating-point, character, pointer, or Boolean.

Switch

A switch statement can evaluate either an integer or a character.

- **Sequence of Execution**

If-else

In the case of 'if-else' statement, either the 'if' block or the 'else' block will be executed based on the condition.

Switch

In the case of the 'switch' statement, one case after another will be executed until the **break** keyword is not found, or the default statement is executed.

- **Default Execution**

If-else

If the condition is not true within the 'if' statement, then by default, the else block statements will be executed.

Switch

If the expression specified within the **switch** statement is not matched with any of the cases, then the default statement, if defined, will be executed.

- **Values**

If-else

Values are based on the condition specified inside the 'if' statement. The value will decide either the 'if' or 'else' block is to be executed.

Switch

In this case, value is decided by the user. Based on the choice of the user, the case will be executed.

- **Use**

If-else

It evaluates a condition to be true or false.

Switch

A **switch** statement compares the value of the variable with multiple cases. If the value is matched with any of the cases, then the block of statements associated with this case will be executed.

- **Editing**

If-else

Editing in 'if-else' statement is not easy as if we remove the 'else' statement, then it will create the havoc.

Switch

Editing in **switch** statement is easier as compared to the 'if-else' statement. If we remove any of the cases from the switch, then it will not interrupt the execution of other cases. Therefore, we can say that the **switch** statement is easy to modify and maintain.

- **Speed**

If-else

If the choices are multiple, then the speed of the execution of 'if-else' statements is slow.

SwitchThe case constants in the switch statement create a jump table at the compile time. This jump table chooses the path of the execution based on the value of the expression. If we have a multiple choice, then the execution of the switch statement will be much faster than the equivalent logic of 'if-else' statement.

Let's summarize the above differences in a tabular form.

	If-else	**switch**
Definition	Depending on the condition in the 'if' statement, 'if' and 'else' blocks are executed.	The user will decide which statement is to be executed.
Expression	It contains either logical or equality expression.	It contains a single expression which can be either a character or integer variable.
Evaluation	It evaluates all types of data, such as integer, floating-point, character or Boolean.	It evaluates either an integer, or character.
Sequence of execution	First, the condition is checked. If the condition is true then 'if' block is executed otherwise 'else' block	It executes one case after another till the break keyword is not found, or the default statement is executed.
Default execution	If the condition is not true, then by default, else block will be executed.	If the value does not match with any case, then by default, default statement is executed.
Editing	Editing is not easy in the 'if-else' statement.	Cases in a switch statement are easy to maintain and modify. Therefore, we can say that the removal or editing of any case will not interrupt the execution of other cases.
Speed	If there are multiple choices	If we have multiple choices then the switch

	implemented through 'if-else', then the speed of the execution will be slow.	statement is the best option as the speed of the execution will be much higher than 'if-else'.

Conclusion:

Thus we had study different types of loop as follows

1) For…..Loop

2) Switch….Cas3

3) Do………While

4) If………Else

And also study **program and design test cases.**

Experiment no. 6

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

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

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

```
 #include <stdio.h>
void main (){

        int i, n=5,j=0; clrscr();
        printf(—enter a no||);


        scanf(—%d||,&i);

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

        }
```

Input **Actual output**

```
 2        2 is even number
          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 2 is even number3 is odd number	Actual output 2 is even number3 is odd number	Remarks success
	4 is even number5 is odd number	4 is even number5 is odd number	
	6 is even number	6 is even number	

Test case no:2

Test case name: Negative values within a range

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

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
123456789122222222222	123456789122222222213	234567891222222215	fail

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

constructObjective

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);
getch();                  printf(—is a odd
}                         number‖); i++;
                      }}j++;
```

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 2 is even number 3 is odd number 4 is even number	Actual output 2 is even number 3 is odd number4 is even number	Remarks success
	5 is odd number 6 is even number	5 is odd number 6 is even number	

Test case no:2

Test case name: Negative values within a range

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

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks

1234567891222222222222	1234567891222222222213	234567891222222215	fail

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

constructObjective

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

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


void main (){

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

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

}
```

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

Test cases:

Test case no: 1

Test case name: Positive values within range

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

Test case no:2

Test case name: Negative values within a range

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

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	234567891222222215	fail

// D. To demonstrate the working of switch

constructObjective

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: Out of range values testing			
Input	**Expected output**	**Actual output s**	**Remark**
Option: 1 a= 22222222222222			
b=22222222222222	44444444444444	-2	fail
Test case no: 3			
Test case name: Divide by zero			
Input	**Expected output**	**Actual output**	**Remarks**
Option: 4			
a= 10 & b=0	error		fail

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

Objective

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

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


void main (){ int
        i;
        clrscr();
        printf(—enter a
        no||);
        scanf(—%d||,&i);

                for(i=1;i<=5;i++){
                        if(i%2==0){

                            printf(—%d||, i);

                            printf(— is a even
                            no||); i++;
                                  }
                      else{
                            printf(—%d||, i);
                            printf(— is a odd
                   no||
                   );
                   i++
                   ;
            }          }
```

Output:

```
}
getch();

        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 number1 is odd number 2 is even number	0 is even number1 is odd number 2 is even number	success

Test case no:2

Test case name: Negative values within a range

Input = -2	Expected output	Actual output	Remarks
	0 is even number	0 is an even number	

	-1 is odd number	-1 is even no	fail
	-2 is even number	-2 is odd no	

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	2345678912222222215	fail

Experiment No.7

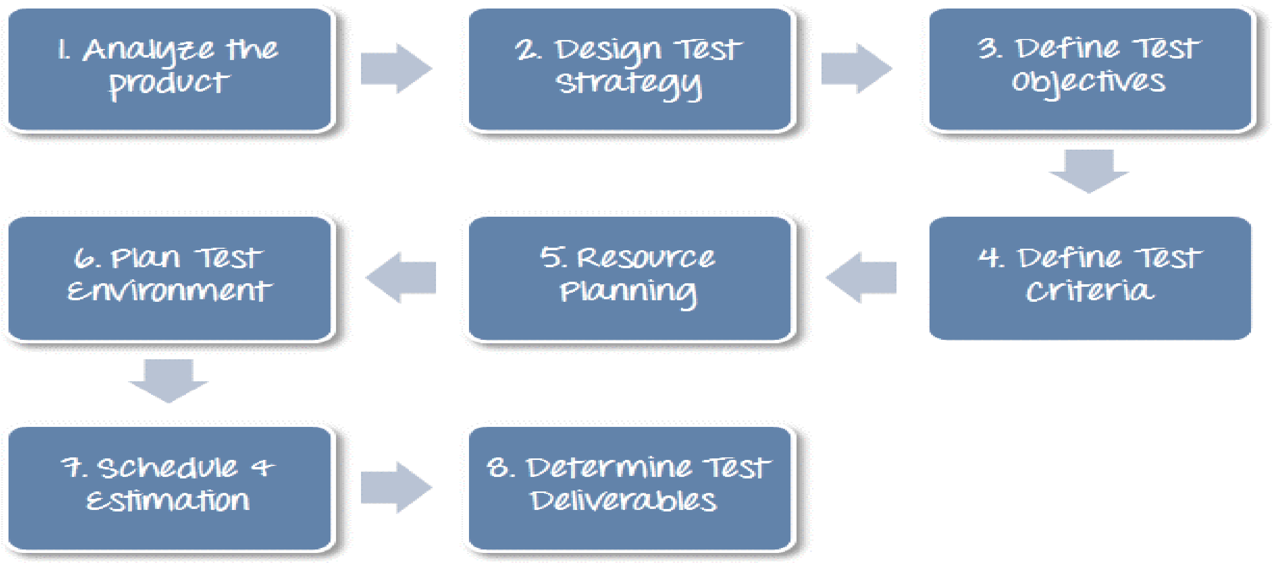
Title: Prepare test plan for an identified mobile/notepad application.

Theory:

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

Test Plan is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1. Analyze the product
2. Design the Test Strategy
3. Define the Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverables



Step 1) Analyze the product

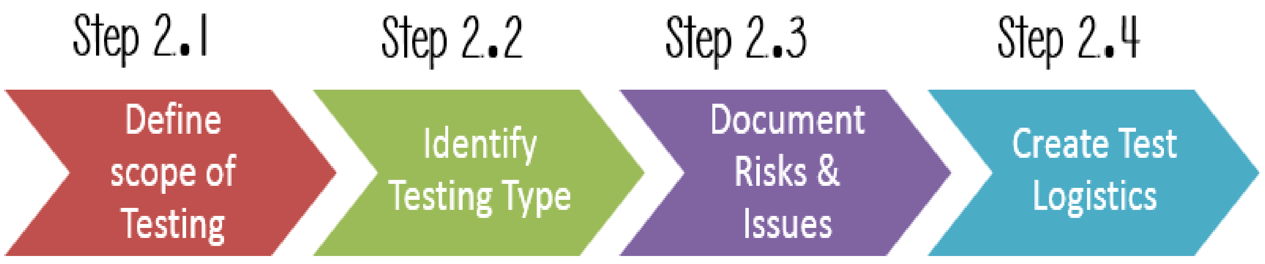
How can you test a product **without** any information about it? The answer is **Impossible.** You must learn a product **thoroughly** before testing it.

Step 2) Develop Test Strategy

Test Strategy is a **critical step** in making a Test Plan in Software Testing. A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:

- The project's **testing objectives** and the means to achieve them
- Determines testing **effort** and **costs**

Back to your project, you need to develop Test Strategy for testing that banking website. You should follow steps below



Step 2.1) Define Scope of Testing

Before the start of any test activity, scope of the testing should be known. You must think hard about it.

- The components of the system to be tested (hardware, software, middleware, etc.) are defined as "**in scope**"
- The components of the system that will not be tested also need to be clearly defined as being "**out of scope**."

Defining the scope of your testing project is very important for all stakeholders. A precise scope helps you

- Give everyone a **confidence & accurate information** of the testing you are doing
- All project members will have a **clear** understanding about what is tested and what is not

How do you determine scope your project?

To determine scope, you must –

- Precise customer requirement
- Project Budget
- Product Specification
- Skills & talent of your test team

Now should clearly define the "in scope" and "out of scope" of the testing.

- As the software requirement specs, the project Guru99 Bank only focus on testing all the **functions** and external interface of website **Guru99** Bank (**in scope** testing)
- Nonfunctional testing such as **stress**, **performance** or **logical database** currently will not be tested. (**out of** scope)

Problem Scenario

The customer wants you to test his API. But the project budget does not permit to do so. In such a case what will you do?

Well, in such case you need to convince the customer that Api Testing is extra work and will consume significant resources. Give him data supporting your facts. Tell him if Api Testing is included in-scope the budget will increase by XYZ amount.

The customer agrees and accordingly the new scopes, out of scope items are

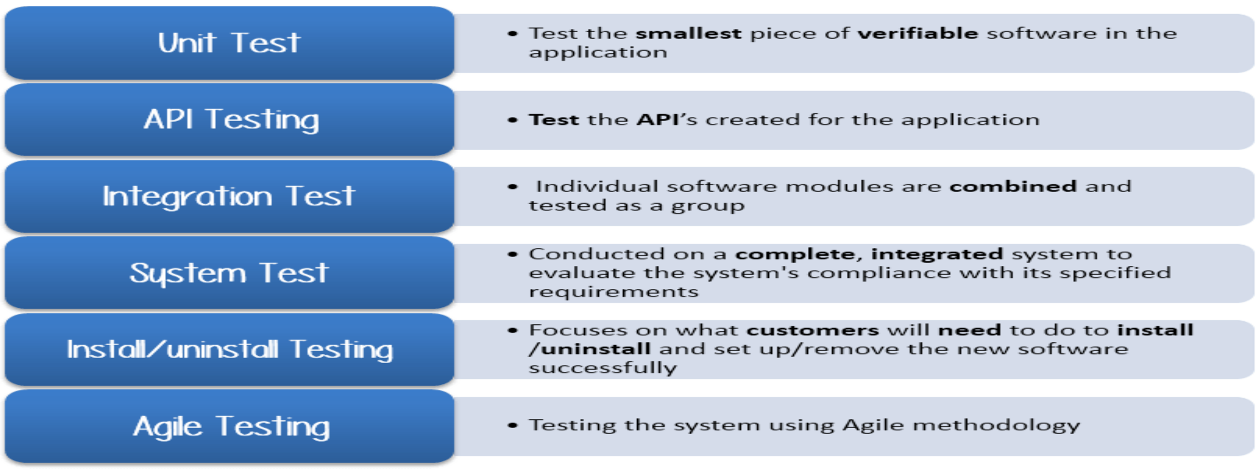
- In-scope items: Functional Testing, Api Testing
- Out of scope items: Database Testing, hardware & any other external interfaces

Step 2.2) Identify Testing Type

A **Testing Type** is a standard test procedure that gives an expected test outcome.

Each testing type is formulated to identify a specific type of product bugs. But, all Testing Types are aimed at achieving one common goal "**Early detection of** all the defects before releasing the product to the customer"

The **commonly used** testing types are described as following figure



Unit Test	• Test the **smallest** piece of **verifiable** software in the application
API Testing	• **Test** the **API**'s created for the application
Integration Test	• Individual software modules are **combined** and tested as a group
System Test	• Conducted on a **complete, integrated** system to evaluate the system's compliance with its specified requirements
Install/uninstall Testing	• Focuses on what **customers** will **need** to do to **install /uninstall** and set up/remove the new software successfully
Agile Testing	• Testing the system using Agile methodology

Commonly Used Testing Types

Step 2.3) Document Risk & Issues

Risk is future's **uncertain event** with a probability of **occurrence** and a **potential** for loss. When the risk actually happens, it becomes the '**issue'.**

In the article Risk Analysis and Solution, you have already learned about the 'Risk' analysis in detail and identified potential risks in the project.

In the QA Test Plan, you will document those risks

Risk	Mitigation
Team member lack the required skills for	Plan **training course** to skill up your members

website testing.	
The project schedule is too tight; it’s hard to complete this project on time	Set **Test Priority** for each of the test activity.
Test Manager has poor management skill	Plan **leadership training** for manager
A lack of cooperation negatively affects your employees’ productivity	**Encourage** each team member in his task, **and inspire** them to greater efforts.
Wrong budget estimate and cost overruns	Establish the **scope** before beginning work, pay a lot of attention to project planning and constantly track and measure the progress

Step 2.4) Create Test Logistics

In Test Logistics, the Test Manager should answer the following questions:

- **Who** will test?
- **When** will the test occur?

Who will test?

You may not know exact names of the tester who will test, but the **type of tester** can be defined.

To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to **fail** or **delay**.

Person having the following skills is most ideal for performing software testing:

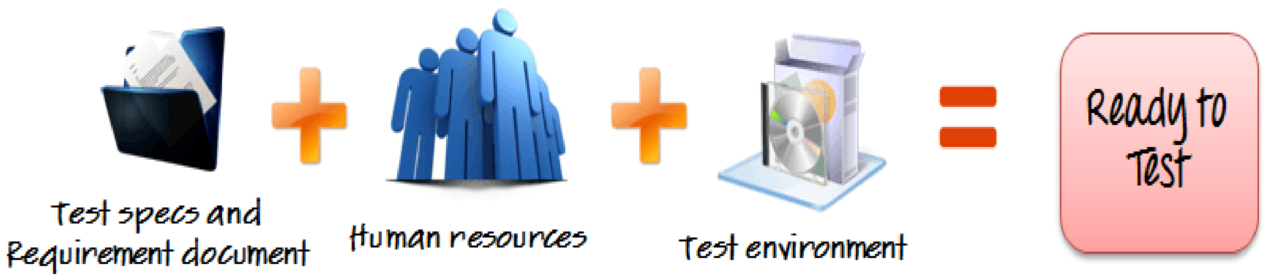
- Ability to **understand** customers point of view
- Strong **desire** for quality
- **Attention** to detail
- Good **cooperation**

In your project, the member who will take in charge for the test execution is the **tester.** Base on the project budget, you can choose in-source or outsource member as the tester.

When will the test occur?

Test activities must be matched with associated development activities.

You will start to test when you have **all required items** shown in following figure



Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is **bug free** before release.

To define the test objectives, you should do 2 following steps

1. List all the software features (functionality, performance, GUI…) which may need to test.
2. Define the **target** or the **goal** of the test based on above features

Let's apply these steps to find the test objective of your Guru99 Bank testing project

You can choose the '**TOP-DOWN'** method to find the website's features which may need to test. In this method, you break down the application under test to **component** and **sub-component**.

- Check that whether website Guru99 **functionality**(Account, Deposit…) is working as expected

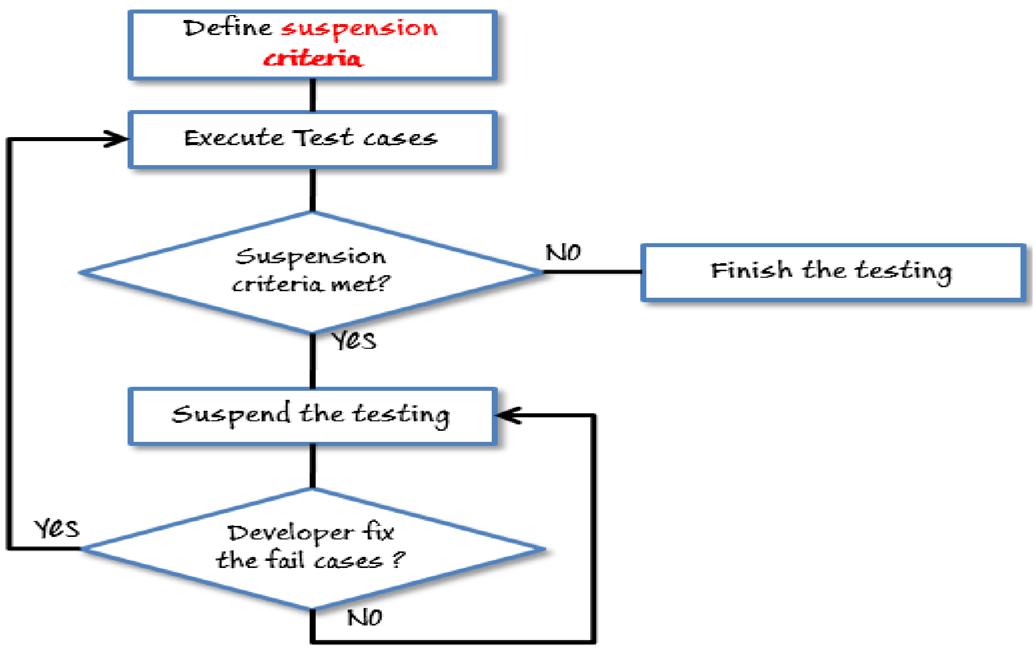
Step 4) Define Test Criteria

Test Criteria is a standard or rule on which a test procedure or test judgment can be based. There're 2 types of test criteria as following

Suspension Criteria

Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be **suspended** until the criteria are **resolved**.

Test Plan Example: If your team members report that there are **40%** of test cases failed, you should **suspend** testing until the development team fixes all the failed cases.



Exit Criteria

It specifies the criteria that denote a **successful** completion of a test phase. The exit criteria are the targeted results of the test and are necessary before proceeding to the next phase of development. Example: **95%** of all critical test cases must pass.

Some methods of defining exit criteria are by specifying a targeted **run rate** and **pass rate**.

- Run rate is ratio between **number test cases executed/total test cases** of test specification. For example, the test specification has total 120 TCs, but the tester only executed 100 TCs, So the run rate is 100/120 = 0.83 (83%)
- Pass rate is ratio between **numbers test cases passed / test cases executed**. For example, in above 100 TCs executed, there're 80 TCs that passed, so the pass rate is 80/100 = 0.8 (80%)

This data can be retrieved in Test Metric documents.

- **Run** rate is mandatory to be **100%** unless a clear reason is given.
- **Pass** rate is dependent on project scope, but **achieving high pass rate** is a goal.

Test Plan Example: Your Team has already done the test executions. They report the test result to you, and they want you to confirm the **Exit Criteria.**

Test Plan for Mobile App Testing

1. Introduction

1.1 Overview

A Test Plan is a document that describes the scope of testing, test strategy, objectives, effort, schedule, and resources required. It serves as a guide to testing throughout the development process. Find an example of the Functional Test Plan here.

1.2 Scope

The scope of work is defined at the beginning of the testing process. A project team should clearly understand what features and functions there are to be tested and which ones are out of scope. To determine the scope of testing, the project specification, budget, and customer's requirements should be taken into account.

1.3 References

This section contains the documents which support the test plan and can be referred to during the testing process. Commonly, the list involves:

- the project plan
- system requirements
- specification design documentation

1.4 Acronyms and Abbreviations

All the acronyms used in the Test Plan require explanation. For example,

- DDD — Database design document
- QMS — Quality Management System
- SLC — Software life cycle

2. Test Plan and Strategy

2.1 Unit Testing

2.1.1 Objective

The main objective of unit testing is to verify whether every unit operates as intended. A function, procedure, method, or even the entire module can be considered a separate unit. Unit testing can be conducted manually, but automated testing is a more common practice.

2.1.2 Entry Criteria

- the planning phase has been finished
- testable units are available
- all functional requirements have been defined
- the unit testing environment has been set up

2.1.3 Exit Criteria

- all planned test cases have been covered
- all the bugs found have been reviewed
- the performance of key modules has been tested

2.1.4 Logging Tests and Reporting

Developers fix the defects found in each unit. If these defects are related to other modules and units, they must be reported.

2.2 System Testing

2.2.1 Objective

System testing is generally conducted after Unit Testing. The objective of System Testing is to evaluate compliance of an integrated application with its requirements.

2.2.2 Testing Procedure

- test cases preparation
- test executions
- bug reporting

2.2.3 Types of System Testing

2.2.3.1 Performance Testing

Performing testing is conducted to detect issues related to:

- memory consumption
- power utilization
- network connectivity
- operating in the background
- switching between applications
- memory leakage

2.2.3.2 Interrupt Testing

This testing type examines how an application reacts to interruption and resumes to its previous state. There are numerous reasons that can potentially interrupt the operation of an app, such as getting a phone call, messages, notifications, battery low, etc.

2.2.3.3 Usability Testing

Usability testing is performed to check whether the application is easy to use and understand for the end-user.

2.2.3.4 Installation and Launch testing

Installation testing aims to detect whether there are any issues during the installation, uninstallation, and updating process. Once the application has been installed, a QA engineer also checks the launching process.

2.2.3.5 Functional Testing

All the functions and features of the application are tested to verify whether they operate according to the specification.

2.2.3.5 Security testing

Security testing is conducted to find the application vulnerabilities and prevent data breaches.

2.2.3.6 Regression testing

Regression testing is a re-execution of tests that had been done before the code changes. Its purpose is to verify whether a new functionality has affected the existing one.

2.3 Pass/Fail Conditions

All the conditions when tests pass or fail are defined and described.

2.5 Test Report

Test Report helps to summarize testing activity in a formal way. It should contain:

- name and overview of an application
- testing hardware and software environment
- the number of test cases executed/passed/failed.

For each issue that has been encountered, the following information is provided:

- bug description
- bug status (open, fixed, etc.)
- bug location
- steps to reproduce an issue

3. Schedules for Testing

A test schedule, created by Project Managers, helps to monitor the testing process.

4. Risks and Assumptions

4.1 Risks

The following risks may occur during the mobile app testing process:

- availability of devices
- new features and modification which have not been planned in advance
- changes in requirements
- delays in schedule

4.2 Assumptions

- each release is accompanied by a note with information about implemented features and their impact on the system
- all blocker bugs receive the high priority status
- all the bugs found are fixed before the next software release
- all documents are up-to-date and delivered to the testing team in time
- all necessary equipment and tools are provided and ready for testing
- the test schedule is reviewed in case there are any obstacles for testing

5. Entry and Exit Criteria

5.1 Entry Criteria

- the development phase has been finished
- requirements have been defined and approved
- test design and tests plan have been created
- the test environment has been set up
- all necessary resources are available.

5.2 Exit Criteria

- tests cases are executed
- the rate of tests cases passed is satisfactory, e.g., 95%
- failed test cases are not related to crucial functionality
- tests results have been accepted
- critical defects have been fixed.

6. Test Metrics

Test metrics are quantitative measures that help to estimate the testing effort. The most common metrics are:

- requirement coverage
- test cases coverage
- number of tests executed
- number of defects found (taking into consideration their priorities and severities)
- tests design effort
- total test effort

7. Logging Tests and Reporting

Each detected issue should be properly reported using special tools and applications.

8. Roles and Responsibilities

Project roles and responsibilities should be clearly defined and divided among the project staff. Commonly, the roles are as follows:

8.1 Project Manager

The Project Manager is responsible for managing the whole testing process. They approve all test documentation, considers budget and time terms, and provide necessary resources.

8.2 Test Lead

The Test Lead is responsible for collecting requirements, planning process, test activity monitoring, and project reporting.

8.3 Test Engineer

The Test Engineer is responsible for test case preparation and execution, as well as issue reporting.

9. Deliverables

The list of testing deliverables usually contains:

- test plan
- test cases documents
- test strategy
- test results
- test summary report

Conclusion: Thus we, had study test plan format and write test plan for mobile app.

Experiment No.8

Title: Prepare defect report after executing test cases for library management system/amount withdrawal from ATM machine/any login form.

Theory:

What is Bug?

A bug is the consequence/outcome of a coding fault.

Defect in Software Testing

A **Defect in Software Testing** is a variation or deviation of the software application from end user's requirements or original business requirements. A software defect is an error in coding which causes incorrect or unexpected results from a software program which does not meet actual requirements. Testers might come across such defects while executing the test cases.
These two terms have very thin line of difference, In the Industry both are faults that need to be fixed and so interchangeably used by some of the Testing teams.

When testers execute the test cases, they might come across such test results which are contradictory to expected results. This variation in test results is referred to as a Software Defect. These defects or variations are referred by different names in different organizations like issues, problems, bugs or incidents.

Bug Report in Software Testing

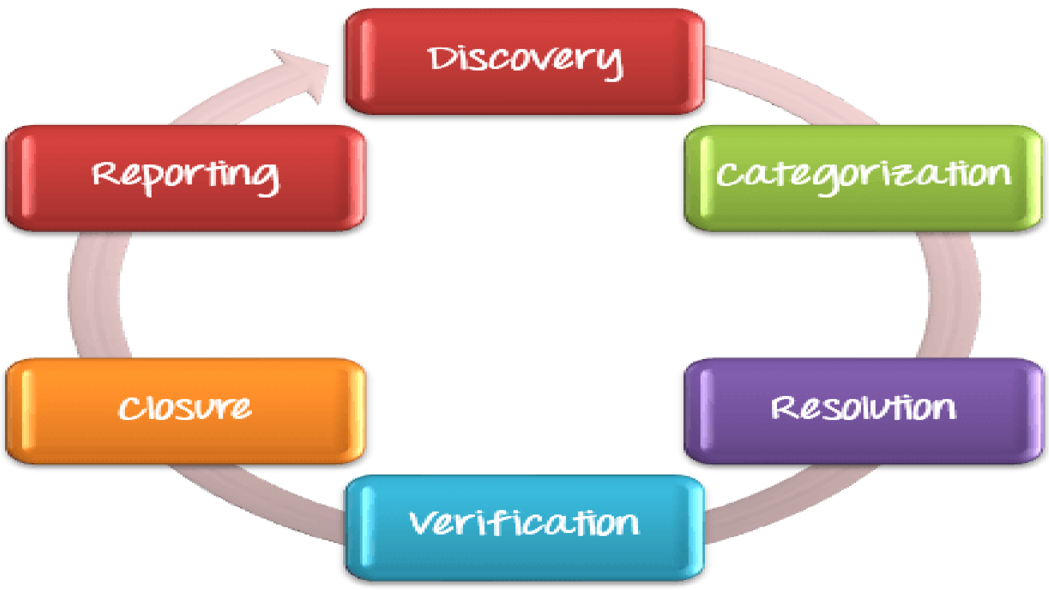
A **Bug Report in Software Testing** is a detailed document about bugs found in the software application. Bug report contains each detail about bugs like description, date when bug was found, name of tester who found it, name of developer who fixed it, etc. Bug report helps to identify similar bugs in future so it can be avoided.
While reporting the bug to developer, your Bug Report should contain the following information

- **Defect_ID** – Unique identification number for the defect.
- **Defect Description** – Detailed description of the Defect including information about the module in which Defect was found.
- **Version** – Version of the application in which defect was found.
- **Steps** – Detailed steps along with screenshots with which the developer can reproduce the defects.
- **Date Raised** – Date when the defect is raised
- **Reference**– where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect
- **Detected By** – Name/ID of the tester who raised the defect
- **Status** – Status of the defect , more on this later
- **Fixed by** – Name/ID of the developer who fixed it
- **Date Closed** – Date when the defect is closed
- **Severity** which describes the impact of the defect on the application

- **Priority** which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively

What is Defect Management Process?

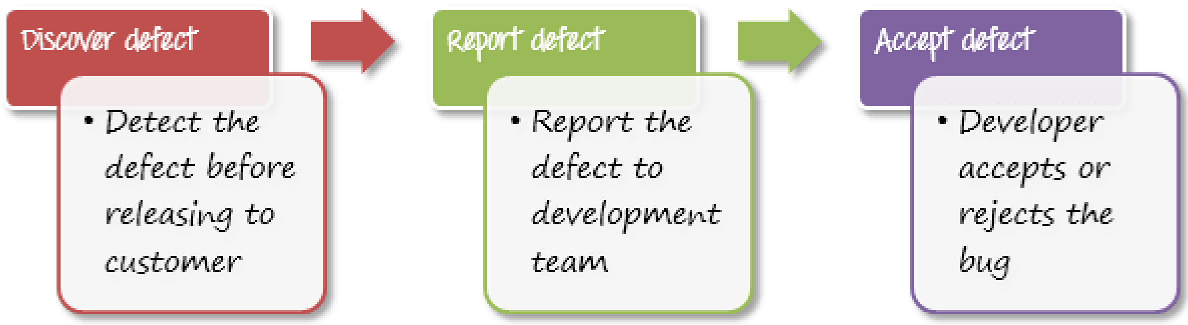
Defect Management is a systematic process to identify and fix bugs. A defect management cycle contains the following stages 1) Discovery of Defect, 2) Defect Categorization 3) Fixing of Defect by developers 4) Verification by Testers, 5) Defect Closure 6) Defect Reports at the end of project
This topic will guide you on how to apply the defect management process to the project Guru99 Bank website. You can follow the below steps to manage defects.



Discovery

In the discovery phase, the project teams have to discover as **many** defects as **possible,** before the end customer can discover it. A defect is said to be discovered and change to status **accepted** when it is acknowledged and accepted by the developers

In the above scenario, the testers discovered 84 defects in the website Guru99.

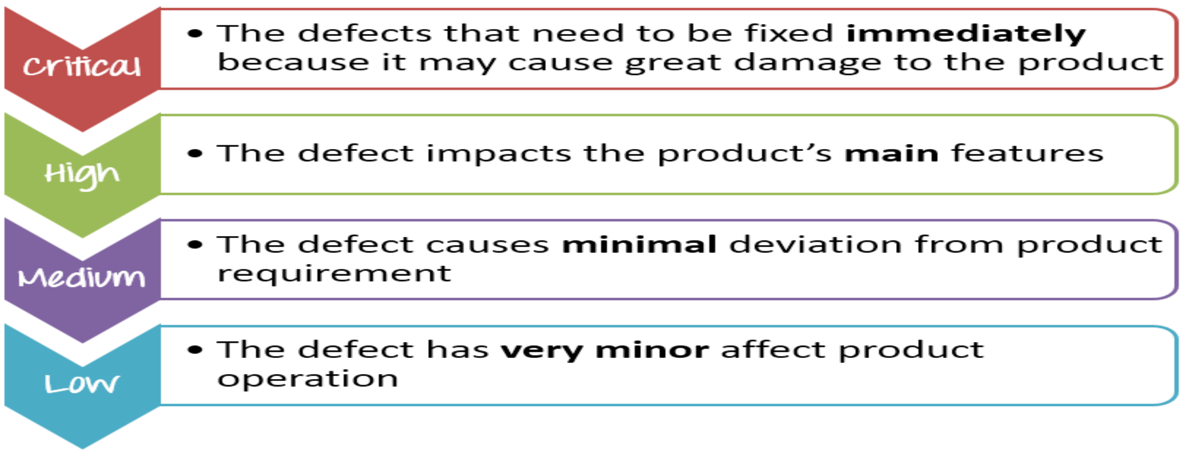

Discover defect
• Detect the defect before releasing to customer
Report defect
• Report the defect to development team
Accept defect
• Developer accepts or rejects the bug

Let’s have a look at the following scenario; your testing team discovered some issues in the Guru99 Bank website. They consider them as defects and reported to the development team, but there is a conflict

the website problem is a defect or not.

Categorization

Defect categorization help the software developers to prioritize their tasks. That means that this kind of priority helps the developers in fixing those defects first that are highly crucial.


Critical
• The defects that need to be fixed **immediately** because it may cause great damage to the product
High
• The defect impacts the product’s **main** features
Medium
• The defect causes **minimal** deviation from product requirement
Low
• The defect has **very minor** affect product operation

Defects are usually categorized by the Test Manager –

Let’s do a small exercise as following

Drag & Drop the Defect Priority Below

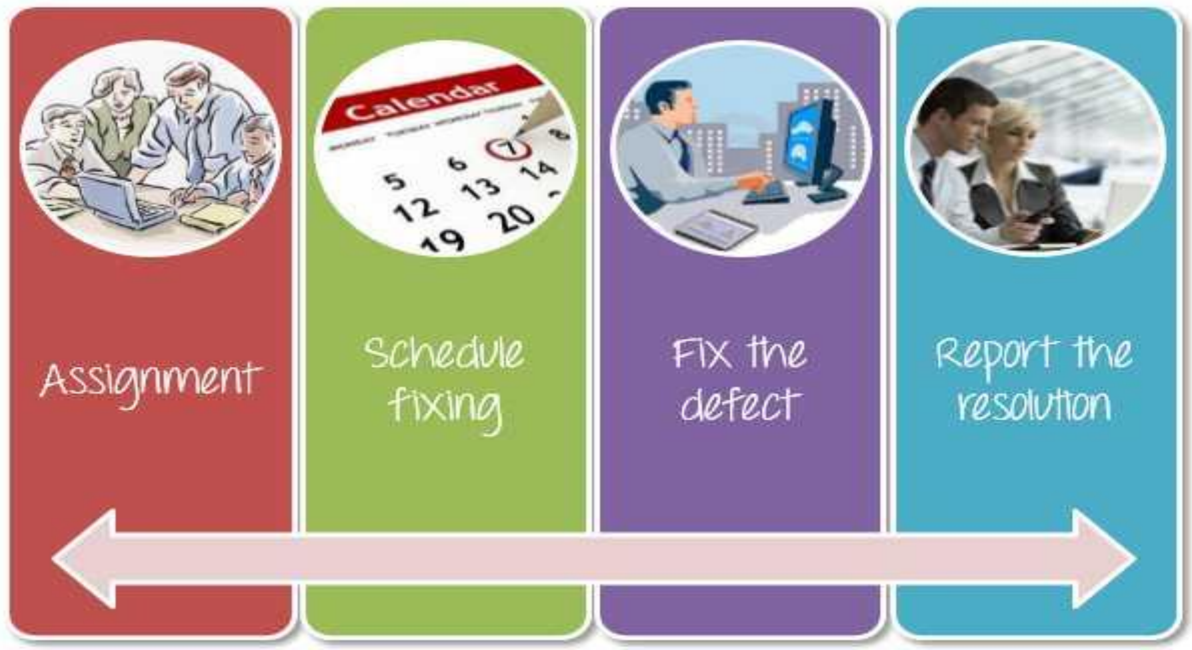
- Critical
- High
- Medium
- Low

1) The website performance is too slow	
2) The login function of the website does not work properly	
3) The GUI of the website does not display correctly on Mobile devices	
4) The website could not remember the user login session	
5) Some links doesn’t work	

No.	Description	Priority	Explanation
1	The website performance is too slow	High	The performance bug can cause huge inconvenience to user.
2	The login function of the website does not work properly	Critical	Login is one of the main function of the banking website if this feature does not work, it is serious bugs
3	The GUI of the website does not display correctly on mobile devices	Medium	The defect affects the user who use Smartphone to view the website.
4	The website could not remember the user login session	High	This is a serious issue since the user will be able to login but not be able to perform any further transactions
5	Some links doesn't work	Low	This is an easy fix for development guys and the user can still access the site without these links

Defect Resolution

Defect Resolution in software testing is a step by step process of fixing the defects. Defect resolution process starts with assigning defects to developers, then developers schedule the defect to be fixed as per priority, then defects are fixed and finally developers send a report of resolution to the test manager. This process helps to fix and track defects easily.
You can follow the following steps to fix the defect.



- **Assignment**: Assigned to a developer or other technician to fix, and changed the status to **Responding**.

- **Schedule fixing**: The developer side take charge in this phase. They will create a schedule to fix these defects, depend on the defect priority.
- **Fix the defect**: While the development team is fixing the defects, the Test Manager tracks the process of fixing defect compare to the above schedule.
- **Report the resolution**: Get a report of the resolution from developers when defects are fixed.

Verification

After the development team **fixed** and **reported** the defect, the testing team **verifies** that the defects are actually resolved.

For example, in the above scenario, when the development team reported that they already fixed 61 defects, your team would test again to verify these defects were actually fixed or not.

Closure

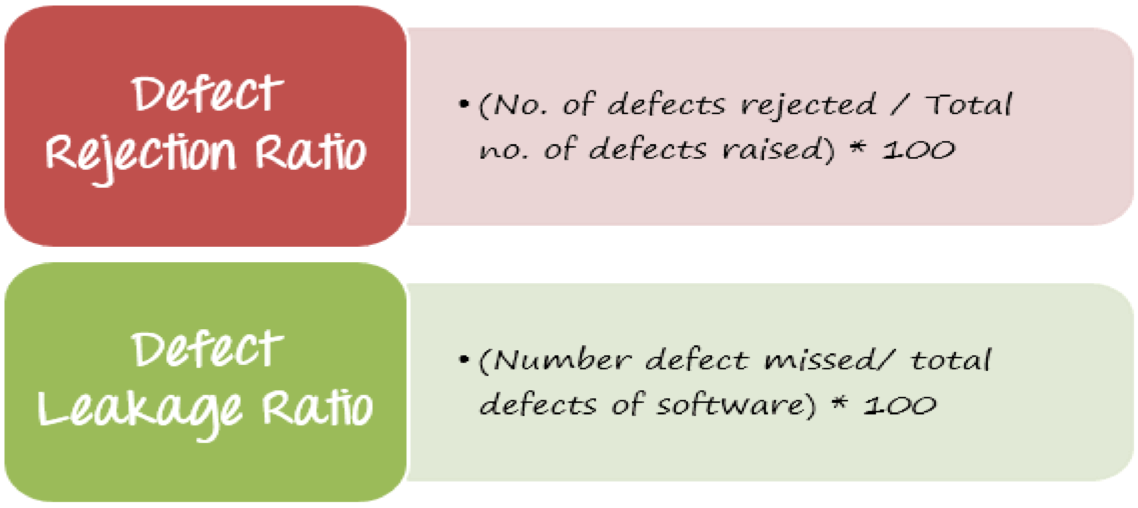
Once a defect has been resolved and verified, the defect is changed status as **closed**. If not, you have send a notice to the development to check the defect again.

Defect Reporting

Defect Reporting in software testing is a process in which test managers prepare and send the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail.
The management board has right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report them the current defect situation to get feedback from them.

How to measure and evaluate the quality of the test execution?

This is a question which every Test Manager wants to know. There are 2 parameters which you can consider as following


Defect Rejection Ratio
• (No. of defects rejected / Total no. of defects raised) * 100
Defect Leakage Ratio
• (Number defect missed/ total defects of software) * 100

In the above scenario, you can calculate the **defection rejection ratio** (DRR) is **20/84 = 0.238 (23.8 %).**

Another example, supposed the Guru99 Bank website has total **64** defects, but your testing team only detect **44** defects i.e. they missed **20** defects. Therefore, you can calculate the defect leakage ratio (DLR) is 20/64 = **0.312** (31.2 %).

Conclusion, the quality of test execution is evaluated via following two parameters

$$\begin{aligned} Defect\ reject\ ratio &= 23.8\% \\ Defect\ leakage\ ratio &= 31.2\% \end{aligned}$$

The smaller value of DRR and DLR is, the better quality of test execution is. What is the ratio range which is **acceptable**? This range could be defined and accepted base in the project target or you may refer the metrics of similar projects.

In this project, the recommended value of acceptable ratio is **5 ~ 10%.** It means the quality of test execution is low. You should find countermeasure to reduce these ratios such as

- **Improve** the testing skills of member.
- **Spend more time** for testing execution, especially for reviewing the test execution results.

Conclusion:

Thus, we had study Defect report format and design defect report for ATM machine.

Content Beyond Syllabus

Experiment No.10

Aim: Study of Any Testing Tool(WinRunner)

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

Importance of Automated Testing:

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

WinRunner Uses:

1. With WinRunner sophisticated automated tests can be created and run on an application.
2. A series of wizards will be provided to the user, and these wizards can create tests in anautomated manner.
3. Another impressive aspect of WinRunner is the ability to record various interactions, and transform them into scripts. WinRunner is designed for testing graphical user interfaces.
4. When the user make an interaction with the GUI, this interaction can be recorded. Recording the interactions allows to determine various bugs that need to be fixed.
5. When the test is completed, WinRunner will provide with detailed information regarding the results. It will show the errors that were found, and it will also give important information about them. The good news about these tests is that they can bereused many times.
6. WinRunner will test the computer program in a way that is very similar to normal user interactions. This is important, because it ensures a high level of accuracy and realism. Even if an engineer is not physically present, the Recover manager will troubleshoot any problems that may occur, and this will allow the tests to be completed without errors.
7. The Recover Manager is a powerful tool that can assist users with various scenarios.This is important, especially when important data needs to be recovered.

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

WinRunner Testing Modes

Context Sensitive

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

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

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

Analog

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

The WinRunner Testing Process

Testing with *WinRunner* involves six main stages:

1. Create the GUI Map

The first stage is to create the GUI map so WinRunner can recognize the GUI objects in the application being tested. Use the RapidTest Script wizard to review the user interface of your application and systematically add descriptions of every GUI object to the GUI map. Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.

2. Create Tests

Next is creation of test scripts by recording, programming, or a combination of both. While recording tests, insert checkpoints where we want to check the response of the application being tested. We can insert checkpoints that check GUI objects, bitmaps, and databases. During this process, WinRunner captures data and saves it as *expected results*—the expected responseof 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 WinRunnerruns a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in **If**, **While**, **Switch**, and **For** statements. We can use the **Syntax Check** options (**Tools >Syntax Check**) to check for these types of syntax errors before running your test.

4. Run Tests

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

5. View Results

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

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

6. Report Defects

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

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

Using Winrunner Window

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

1.4.1. To start WinRunner:

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

The first time you start WinRunner, the Welcome to WinRunner window and the —What‘s New in WinRunner‖ help open. From the Welcome window you can create a new test, open an existing test, or view an overview of WinRunner in your default browser.
CASE TOOLS & SOFTWARE TESTING LAB MANUAL

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

1.4.2. The Main WinRunner Window

The main WinRunner window contains the following key elements:

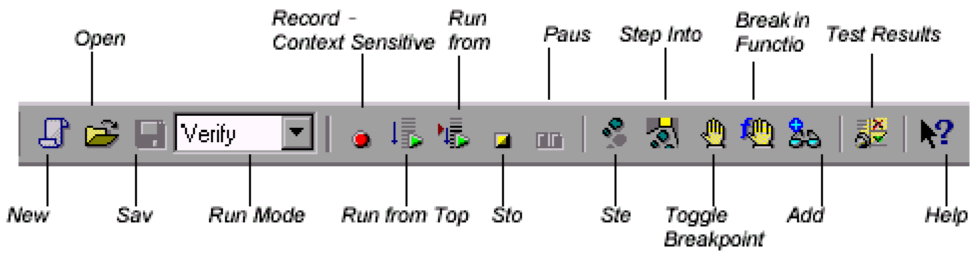
WinRunner title bar

Menu bar, with drop-down menus of WinRunner commands

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

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

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



Standard Toolbar

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

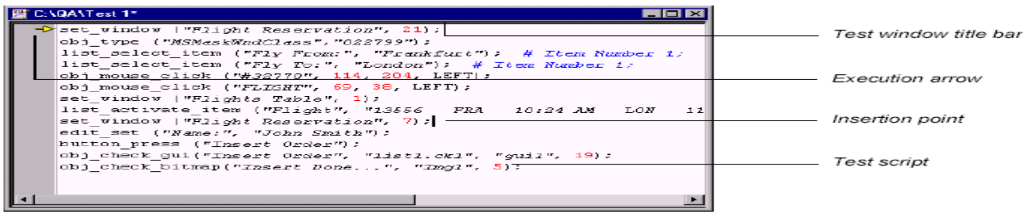
Note that you can also execute many commands using *softkeys*. Softkeys are keyboard shortcuts for carrying out menu commands. You can configure the softkey combinations for your keyboard using the Softkey Configuration utility in your WinRunner program group. For more information, see the ―WinRunner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

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

The Test Window

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

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



```
set_window ("Flight Reservation", 21);
obj_type ("MSMaskWndClass","022799");
list_select_item ("Fly From:", "Frankfurt");  # Item Number 1;
list_select_item ("Fly To:", "London");  # Item Number 1;
obj_mouse_click ("#32770", 114, 204, LEFT);
obj_mouse_click ("FLIGHT", 69, 38, LEFT);
set_window ("Flights Table", 1);
list_activate_item ("Flight", "13556   FRA   10:24 AM   LON   11
set_window ("Flight Reservation", 7);
edit_set ("Name:", "John Smith");
button_press ("Insert Order");
obj_check_gui("Insert Order", "list1.ckl", "gui1", 19);
obj_check_bitmap("Insert Done...", "Img1", 5);
```

Experiment 10

Title : Study of any web testing tool (e.g. Selenium)

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

One of Selenium's key features is the support for executing one's tests on multiple browser platforms.

Selenium Components

Selenium is composed of three major tools. Each one has a specific role in aidingthe development of web application test automation.

Selenium-RC provides an API (Application Programming Interface) and library for eachof 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 suitesthat 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 onlyas long to run as the longest individual test.

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

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

Flexibility and Extensibility

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

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

.Test Suites

A test suite is a collection of tests. Often one will run all the tests in a test suite as one continuous batch-job.

- When using Selenium-IDE, test suites also can be defined using a simple HTML file.The syntax again is simple. An HTML table defines a list of tests where each row defines the filesystem path to each test. An example tells it all.

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

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, ispresent on the page.

verifyText – verifies expected text and it‘s corresponding HTML tag are present on thepage.
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 byits HTML tag, is present on the page.

Experiment no 10

Aim: Study of Any Bug Tracking Tool (Bugzilla)

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

Advantage of Using Bugzilla:

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

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

The bugzilla utility basically allows to do the following:

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

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

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

Using Bugzilla:

Bugzilla usage involves the following activities

Setting Parameters and Default Preferences

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

Setting Parameters and Default Preferences:

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

▪ Set the *maintainer*

▪ Set the *mail_delivery_method*

▪ Set *bug change policies*

▪ Set the display order of bug reports

To set parameters and default preferences:

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

Creating a New User

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

To create a new user:

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

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

Impersonating a User

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

Adding Products

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

<u>To add a product:</u>

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

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

Adding Product Components

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

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

To add a component:

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

Modifying Default Field Values

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

To modify default field values:

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

Creating a New Bug

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

To create a new bug:

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

2. If we‘ve defined more than one component, choose the component from the component

list.

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

Viewing Bug Reports

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

To find a bug:

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

Modifying Bug Reports

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

To modify a bug report:

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

Experiment no10

Aim: Study of Any Test Management Tool (Test Director)

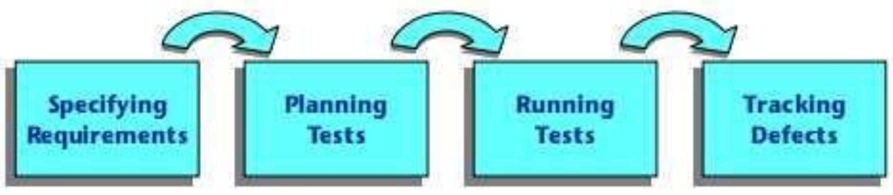
Test Director is a global test management solution which provides communication, organization, documentation and structure to the testing project.

Test Director is used for

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

The TestDirector Testing Process

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



Specifying Requirements

- Requirements are linked to tests and defects to provide complete traceability and aid thedecision-making process
- See what percent of requirements are covered by tests
- Each requirement in the tree is described in detail, and can include any relevant attachments. The QA tester assigns the requirement a priority level which is taken into consideration when the test team creates the test plan
- Import from Microsoft Word or third party RM tool

Planning Tests

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

Define subjects according to:

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

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

Running Tests

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

By creating test sets and choosing which tests to include in each set, test suite can be created. A test set is a group of tests in a TestDirector project database designed to achieve specific testing goals.

Tests can be run manually or scheduled to run automatically based on application dependencies.

Tracking Defects

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

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

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

System Requirements for TestDirector

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

<u>Experiment 10</u>

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

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

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

Overall structure

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

Products and Test Plans

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

Product: A Product is something that will exist forever in TestLink. Products willundergo many different versions throughout their life times. Product includes Test Specification with Test Cases and should be sorted via Keywords.

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

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

Test Case Categorization

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

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

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

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

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

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

Test Specification

Creating Test Cases

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

User can also copy or move Test Cases.

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

Deleting Test Cases

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

Requirements relation

Test cases could be related with software/system requirements as n to n. The functionality must be enabled for a Product. User can assign Test Cases and Requirements via link Assign Requirements in the main screen.

Test Plans

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

Creating a new Test Plan

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

Test Execution

Test execution is available when:

1. A Test Specification is written.

2. A Test Plan is created.

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

4. A Build is created.

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

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

Test Status

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

Insert Test results

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

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

Advantages:

1. Easy in tracking test cases(search with keyword, test case id, version etc)
2. We can add our custom fields to test cases.

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

Example of TestLink workflow:

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

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

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

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

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

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

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

8. Manager would like to see results. Administrator explains him that he can create account himself on the login page. Manager does it. He has ―Guest‖ rights and could see results and Test cases. He can see that everything passed in overall report and problems in build ―Fish 0.1‖ in a report for particular Build.

Assignment No.10

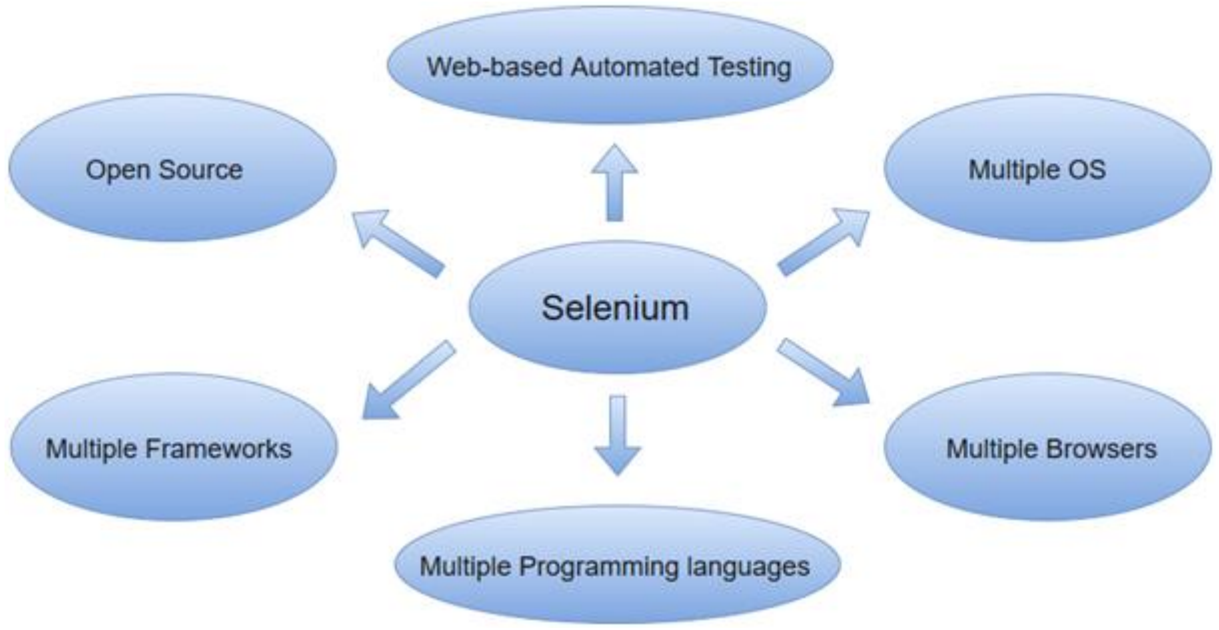
Aim:- Automate the test cases using Selenium.

Theory:

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite.It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language.Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven**, **Jenkins**, **& Docker** to achieve continuous testing. It can also be integrated with tools such as **TestNG**, & **JUnit** for managing test cases and generating reports.

Selenium Features

- Selenium is an open source and portable Web testing Framework.
- Selenium IDE provides a playback and record feature for authoring tests without the need to learn a test scripting language.
- It can be considered as the leading cloud-based testing platform which helps testers to record their actions and export them as a reusable script with a simple-to-understand and easy-to-use interface.

- Selenium supports various operating systems, browsers and programming languages. Following is the list:
  - Programming Languages: C#, Java, Python, PHP, Ruby, Perl, and JavaScript
  - Operating Systems: Android, iOS, Windows, Linux, Mac, Solaris.
  - Browsers: Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- It also supports parallel test execution which reduces time and increases the efficiency of tests.
- Selenium can be integrated with frameworks like Ant and Maven for source code compilation.
- Selenium can also be integrated with testing frameworks like TestNG for application testing and generating reports.
- Selenium requires fewer resources as compared to other automation test tools.
- WebDriver API has been indulged in selenium whichis one of the most important modifications done to selenium.
- Selenium web driver does not require server installation, test scripts interact directly with the browser.
- Selenium commands are categorized in terms of different classes which make it easier to understand and implement.
- Selenium Remote Control (RC) in conjunction with WebDriver API is known as Selenium 2.0. This version was built to support the vibrant web pages and Ajax.

Code

Input java script:

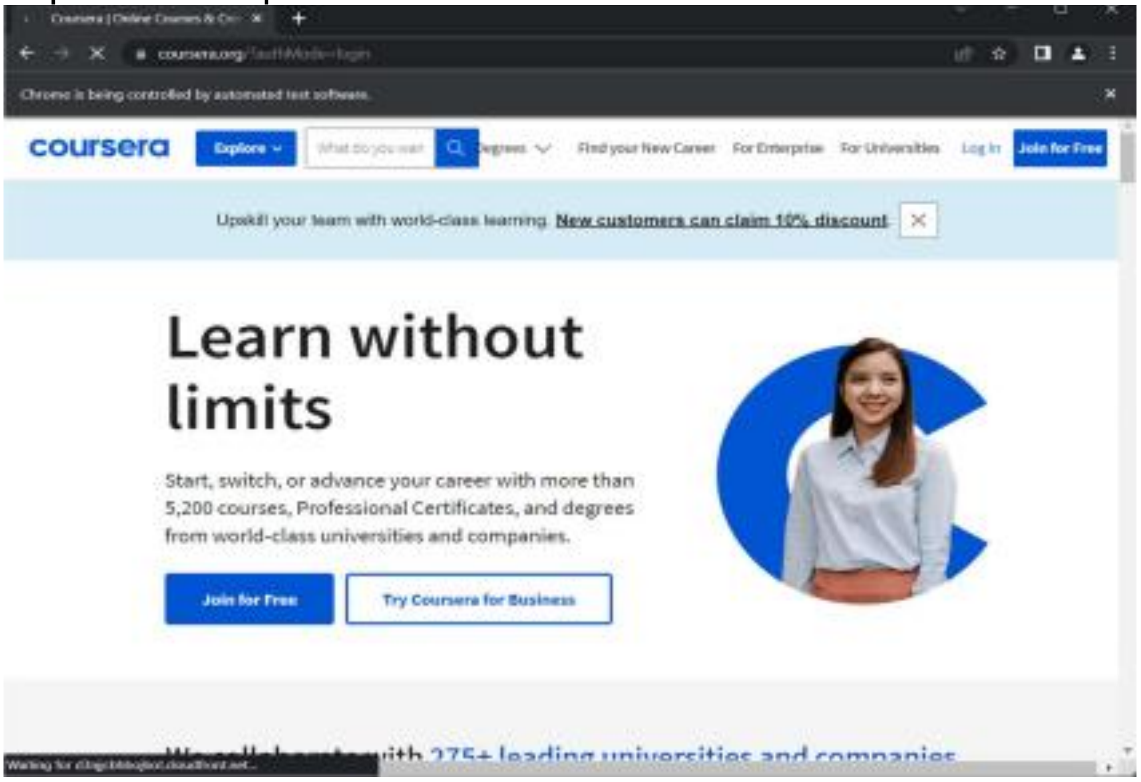
```
package demo;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;
public class SeleniumDemo {
 public static void main(String[] args) {
 WebDriverManager.chromedriver().setup();
 WebDriver driver = new ChromeDriver();
 driver.get("https://www.coursera.org/?authMode=login");
driver.findElement(By.id("email")).sendKeys("abc@gmail.com");
driver.findElement(By.id("password")).sendKeys("thisispassword");

driver.findElement(By.xpath("//button[@class='_6dgzsvq css-1af0gyj']")).click();   driver.close();

 }

}
```

Pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>

 <groupId>SeleniumDemo</groupId>

 <artifactId>SeleniumDemo</artifactId>

 <version>0.0.1-SNAPSHOT</version>

 <build>

 <sourceDirectory>src</sourceDirectory>

 <plugins>

 <plugin>

 <artifactId>maven-compiler-plugin</artifactId>

 <version>3.8.1</version>

 <configuration>

 <source>1.7</source>

 <target>1.7</target>

 </configuration>

 </plugin>

 </plugins>

 </build>

 <dependencies>

 <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->   <dependency>

 <groupId>org.seleniumhq.selenium</groupId>

 <artifactId>selenium-java</artifactId>
```

```
<version>4.1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->   <dependency>

<groupId>io.github.bonigarcia</groupId>
<artifactId>webdrivermanager</artifactId>
<version>5.3.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->   <dependency>

<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>2.11.0</version>
</dependency>
</dependencies>
</project>
```

- **Output of above script:**



Chrome is being controlled by automated test software.

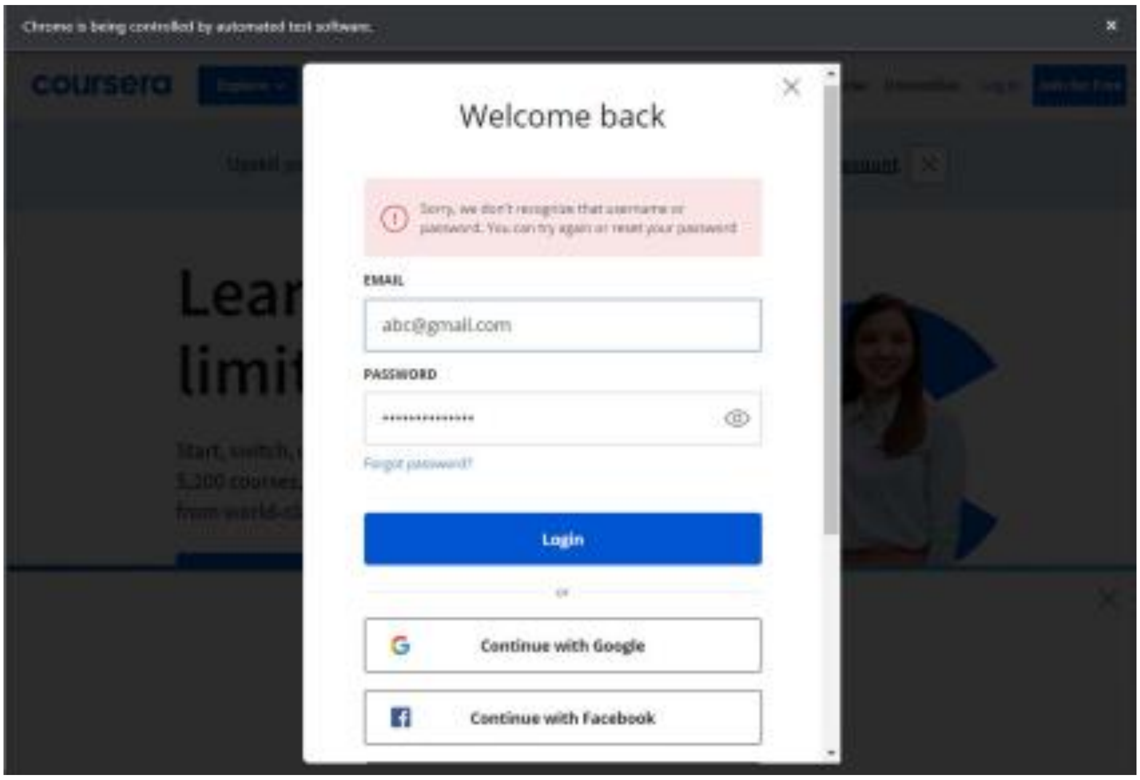
Upskill your team with world-class learning. New customers can claim 10% discount.

Learn without limits

Start, switch, or advance your career with more than 5,200 courses, Professional Certificates, and degrees from world-class universities and companies.

Coursera | Online Courses & Cr
coursera.org/?authMode=login
coursera
Explore
What do you want
Degrees
Find your New Career
For Enterprise
For Universities
Log In
Join for Free
Join for Free
Try Coursera for Business
We collaborate with 275+ leading universities and companies

-


Chrome is being controlled by automated test software.

Welcome back

Sorry, we don't recognize that username or password. You can try again or reset your password

EMAIL: abc@gmail.com

PASSWORD: •••••••••••••••

Conclusion: Hence we have studied the Automation by using Selenium.