INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

SOFTWARE TESTING METHODOLOGIES LABORATORY								
VI Semester: IT								
Course Code	Category	Hours /Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AITC23	Core	1	-	2	2	30	70	100
Contact Classes:12	Tutorial Classes: Nil	Practical Classes: 33				Total Classes: 45		
Prerequisite: Object Oriented Software Engineering Laboratory								

I. COURSE OVERVIEW:
Software testing improves the quality of software and provides error free software. This course provides the hands-on experience on an automated and manual testing techniques such as black-box, data flow, path and transaction testing. The demonstration on bug tracking and automated test management tools. Testing is major part in software development life cycle.

II. COURSES OBJECTIVES:
The students will try to learn:

I. The software testing and quality assurance as a fundamental component of software life cycle.
II. Describe fundamental concepts of software quality assurance.
III. How to use software quality tools and analyze their effectiveness.
IV. The quality management, assurance, and quality standard to software system.

III. COURSE OUTCOMES:

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

CO1	Demonstrate the power of wide variety of testing techniques in developing qualitative software as per customer needs
CO2	Make use of various system testing strategies at various levels for analyzing likelihood of faults and generating defect free software product.
CO3	Utilize Testing plans and procedures for developing effective software product.
CO4	Analyze automated Testing models for evaluating correctness of real time software systems.
CO5	Illustrate the importance of standards in the quality management process and their impact on the final product.
CO6	Inspect Quality assurance tools techniques to manage Risk and assess quality of software developed for engineering applications

IV. COURSE CONTENT:

SOFTWARE TESTING METHODOLOGIES LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises:

Introduction:

Software testing is a crucial aspect of software development, aimed at ensuring the quality, reliability, and correctness of software products. The Software Testing Methodologies Laboratory provides hands-on experience to students in various software testing techniques, tools, and methodologies. This lab is designed to complement theoretical knowledge with practical skills necessary for identifying and rectifying defects in software systems.

1.1 Different range of values and test cases (do while)

Analyse the working of do while with different range of values and test cases.

Hint

Program:

```
do
{
if(-----)
{
printf("%d", i);
printf("is a even no.");
i++;
else
}
while(-----);
```

Test Cases:

```
Test Case 1: Positive Range

Description: Test the do-while loop with a positive range of values.
Test Input: Start = 1, End = 5
Expected Output: The loop should execute 5 times, printing numbers from 1 to 5.
```

1.2 Different range of values and test cases (while)

Analyse the working of while with different range of values and test cases.

Hint

Program:

```
while(----)
{
if(----)
{
printf(“%d”,i);
printf(“is a even number”);
i++;
```

```
j++;
}
else{
printf(“%d”,i);
printf(“is a odd number”);
i++;
j++; }
```

Test Cases:

```
Test Case 1: Positive Range

Description: Test the while loop with a positive range of values.
Test Input: Start = 1, End = 5
Expected Output: The loop should execute 5 times, printing numbers from 1 to 5.

Test Case 2: Negative Range

Description: Test the while loop with a negative range of values.
Test Input: Start = -3, End = -1
Expected Output: The loop should execute 3 times, printing numbers from -3 to -
1.
```

1.3 Different range of values and test cases (if else)

Analyse the working of if else with different range of values and test cases.

Hint

Program:

```
if( ---)
{
printf("number is even no:%d\n",i);
}
else
printf("number is odd no:%d\n",i);
}
}
```

Test Cases:

```
Test Case 1: Positive Value

Description: Test the if-else construct with a positive value.
Test Input: Value = 5
Expected Output: If the value is greater than 0, print "Positive"; otherwise,
print "Non-positive".

Test Case 2: Negative Value

Description: Test the if-else construct with a negative value.
Test Input: Value = -3
Expected Output: If the value is less than 0, print "Negative"; otherwise, print
"Non-negative".
```

1.4 Different range of values and test cases (switch)

Analyse the working of switch with different range of values and test cases.

Hint

Program:

```
switch(---) {
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;
```

Test Cases:

```
Test Case 1: Single Character

Description: Test the switch construct with a single character input.
Test Input: Character = 'a'
Expected Output: Depending on the character input, execute the corresponding case
statement.
Test Case 2: Integer Value

Description: Test the switch construct with an integer input.
Test Input: Value = 5
Expected Output: Depending on the integer input, execute the corresponding case
statement.
```

1.4 Different range of values and test cases(for)

Analyse the working of for with different range of values and test cases.

Hint

Program:

```
for(-----)
{
if(----)
{
printf(“%d”, i);
printf(“ is a even no”);
i++;
}
printf(“%d”, i);
printf(“ is a odd no”); i++;
}
```

Test Cases:

```
Test Case 1: Positive Range

Description: Test the for loop with a positive range of values.
Test Input: Start = 1, End = 5
Expected Output: The loop should iterate from 1 to 5 inclusive.
Test Case 2: Negative Range

Description: Test the for loop with a negative range of values.
Test Input: Start = -3, End = -1
Expected Output: The loop should iterate from -3 to -1 inclusive.
```

Try:

Design and execute unit test to ensure its correctness of the factorial of a given number.

Hint

```
Unit tests
    assert(factorial(0) == 1);  // Test factorial of 0
    assert(factorial(1) == 1);  // Test factorial of 1
    assert(factorial(5) == 120);  // Test factorial of 5
```

2 System Specifications and Report the Bugs.

2.1 ATM System

Design and develop a program for ATM System by using any suitable programming language.

Hint

Program:

```
switch (-----)
{
case 1:
printf("\n YOUR BALANCE =Rs.%lu ", amount);
break;
case 2:
printf("\n ENTER THE AMOUNT: ");
scanf("%lu", &withdraw);
if (------)
{
--------
case 3:
printf("\n ENTER THE AMOUNT: ");
scanf("%lu", &deposit); amount = amount + deposit;
printf(" YOUR BALANCE =RS.%lu", amount);
break;
case 4:
printf("\n THANK YOU USING OUR ATM SERVICES");
break; default:
printf("\n INVALID CHOICE");
}}
```

Test Cases:

```
System Specifications:

Cash Withdrawal:

Users should be able to withdraw cash from their accounts.
The system should ensure that withdrawal limits are enforced.
The system should dispense the correct amount of cash.

Cash Deposit:

Users should be able to deposit cash into their accounts.
The system should verify the authenticity of deposited cash.
The system should update the account balance accurately after a deposit.
```

```
Common Bugs:
Authentication Flaws:

Weak or improper authentication mechanisms may lead to unauthorized access.
Bugs related to PIN verification can result in security vulnerabilities.
Account Balance Errors:

Incorrect calculation or display of account balances.
Failures in updating account balances after transactions.
```

2.2 Banking Application

Design and develop a program for banking application by using any suitable programming language.

Hint

Program:

```
switch(-----)
{
case '1': Create_new_account();
break;
case '2': Cash_Deposit(); break;
case '3': Cash_withdrawl(); break;
case '4': Account_information(); break;
case '5': return 0;
case '6': system("cls");
break;
}
```

Test Cases:

```
Account Balance:

Test Case 1: Verify that the user can view their account balance after successful
login.
Test Case 2: Verify that the displayed account balance is accurate and matches
the actual balance.
Test Case 3: Verify that the account balance is updated accurately after
transactions (withdrawals, deposits, transfers).
```

```
Cash Withdrawal:

Test Case 4: Verify that the user can withdraw cash within their available balance.
Test Case 5: Verify that the system dispenses the correct amount of cash requested.
Test Case 6: Verify that the system updates the account balance correctly after a withdrawal.
Test Case 7: Verify that the user receives a receipt for the withdrawal transaction.
```

2.3 Student records into table into Excel file

Design and develop a program to update 10 student records into table into Excel file by using any suitable programming language.

Hint

Program:

```
for (----------)
{
a[i][j] = ------();s.getCell(j, i).getContents();
int x= Integer.parseInt(a[i][j]);
if(x > 35)
{
Label l1 = new Label (6, i, "pass");
ws.addCell(l1);
}
else
{
Label l1 = new Label (6, i, "fail");
ws.addCell(l1);
break;
}
}
```

2.4 Calculator

Design and develop representing a calculator in any suitable programming language and write down its test cases.

Hint

Test Cases:

```
Test Cases:
Test addition: Verify that the addition method returns the correct sum for two positive numbers.
Test subtraction: Verify that the subtraction method returns the correct difference for two positive numbers.
```

Try:

1.Find the maximum element in an array. Design black-box tests to validate the functionality without looking at the implementation details.

Hint

```
// Black-box tests
void testFindMax() {
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {5, 4, 3, 2, 1};
    int arr3[] = {3, 8, 2, 6, 9, 1, 7};
    int arr4[] = {-5, -10, -3, -7, -2};
    int arr5[] = {100};
```

3. Test Cases:

3.1 Test Cases For GMAIL

Create test cases for Gmail involves covering various functionalities and scenarios to ensure the proper functioning of the Gmail service.

Hint

Test Cases:

```
Test Case Title: Verify Login Functionality

Description: This test case verifies the login functionality of the Gmail application.
Steps to Reproduce:
Launch the Gmail application.
Enter a valid username (email address) in the provided field.
Enter the corresponding password in the password field.
Click on the "Sign in" button.
Expected Result: The user should be successfully logged into their Gmail account and directed to the inbox.
```

3.2 Test Cases For FACEBOOK.

Create test cases for Facebook involves covering various functionalities of the platform, considering factors like usability, security, and compatibility.

Hint

Test Cases:

```
Test Case: Verify that users can successfully register for a new account on Facebook.
Steps:
Navigate to the Facebook registration page.
Enter valid information into the required fields (e.g., name, email, password, date of birth).
Click on the "Sign Up" button.
Validate that the user is directed to the account verification page or home feed.
Expected Result: The user should be able to register for a new account without encountering any errors.
```

3.3 Test Cases For TWITTER.

Creating comprehensive test cases for Twitter involves covering various aspects of the platform, including user interactions, content sharing, security, and performance.

Hint

Test Cases:

```
Test Case: Validate that registered users can log in to their Twitter accounts.
Steps:
Access the Twitter login page.
Enter valid credentials (email/phone and password).
Click on the "Log In" button.
Verify that the user is logged in and redirected to the Twitter home feed.
Expected Result: The user should be able to log in successfully and access their
account.
```

Try:

1. Develop a simple Java program and write unit tests using JUnit to cover different functionalities.

4. Testing Technique: Boundary Value Analysis.

Boundary Value Analysis (BVA) is a software testing technique used to identify errors at the boundaries of input domains rather than focusing on the middle values. It is based on the principle that errors often occur at the boundaries of input ranges rather than within the range itself. The idea is to test the minimum, maximum, and just beyond the minimum and maximum values of valid input ranges.

4.1 Triangle Problem:

Design and develop a program in any suitable programming language of your choice to solve the triangle problem. defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, execute the test cases and discuss the results.

Hint

Program:

```
if(------)
{
printf("value of a is out of range"); if(!c2)
printf("value of b is out of range");
if(---)
printf("value of c  is out of range");
}while(!c1 || !c2 || !c3); if((a+b)>c && (b+c)>a && (c+a)>b)
{
if(a==b && b==c)
printf("Triangle is equilateral\n"); else if(a!=b && b!=c && c!=a)
printf("Triangle is scalene\n"); else
printf("Triangle is isosceles\n");
}
else
```

```
printf("Triangle cannot be formed \n"); }
```

Test Cases:

```
Test cases for Triangle:
Equilateral Triangle:

Test Case 1: All sides are equal.
Input: Side lengths - 5, 5, 5
Expected Output: Equilateral
Isosceles Triangle:

Test Case 2: Two sides are equal.
Input: Side lengths - 4, 4, 6
Expected Output: Isosceles
```

4.2 Next Date Program:

Design, develop, code and run the program in any suitable programming language to implement the Next Date function. Analyze it from the perspective of boundary value testing, derive different test cases, execute these test cases and discuss the test results.

Hint

Program:

```
if(-----)
else
return 0;
}
int main()
{
int day,month,year,tomm_day,tomm_month,tomm_year; char flag;
do
{
flag='y';
printf("\nenter the today's date in the form of dd mm yyyy\n");
scanf("%d%d%d",&day,&month,&year);
tomm_month=month; tomm_year= year; if(day<1 || day>31)
{
printf("value of day, not in the range 1...31\n"); flag='n';
}
if(------)
{
printf("value of month, not in the range 1.       12\n");
flag='n';
}
else if(check(day,month))
{
printf("value of day, not in the range day<=30"); flag='n';
}
if(------)
{
printf("value of year, not in the range 1812.     2015\n");
flag='n';
}
if(-----)
{
if(------)
```

```
{
printf("invalid date input for leap year"); flag='n';
}
else if(!(isleap(year))&& day>28)
{
printf("invalid date input for not a leap year"); flag='n';
}
}
}while(-----);
```

Test cases:

```
Input: 31st December 2021
Expected Output: 1st January 2022
```

4.3 Commission Problem:

Design, develop, code and run the program in any suitable language to solve the commission problem. Analyze it from the perspective of boundary value testing, derive different test cases, execute these test cases and discuss the test results.

Hint

Program:

```
if(-----)
printf("\nValue of locks are not in the range of 1.70\n");
else
{
temp=totallocks+locks; if(temp>70)
printf("New totallocks = %d not in the range of 1.70\n",temp);
else
totallocks=temp;
}
printf("Total locks = %d",totallocks); if(c2)
printf("\n Value of stocks not in the range of 1. 80\n");
else
{
temp=totalstocks+stocks;
 if(-----)
printf("\nNew total stocks = %d not in the range of 1.80",temp);
else
totalstocks=temp;
}
printf("\nTotal stocks = %d",totalstocks); if(c3)
printf("\n Value of barrels not in the range of 1.90\n");
else
{
temp=totalbarrels+barrels; if(temp>90)
printf("\nNew total barrels = %d not in the range of 1.90\n",temp);
else
totalbarrels=temp;
}
```

Try:

1. Compare and contrast boundary value analysis and equivalence partitioning as test case design techniques. Which one would be more suitable for testing a given scenario, and why?

5.Testing Technique: Equivalence Class Testing

Equivalence class testing is a software testing technique used to reduce the number of test cases while still effectively testing the software. It involves partitioning the input data of a software component into groups or classes such that each class can be considered as equivalent from the perspective of the system's behaviour. By testing representative values from each equivalence class, you can ensure that the software behaves consistently across each class.

5.1 Web Application that requires a username:

Design and develop a login form for a web application that requires a username using Equivalence class Testing.

Hint

```
Equivalence classes for Username:

Class 1: Valid usernames (length 5-15 characters, alphanumeric)
Class 2: Invalid usernames (length < 5)
```

Test Cases:

```
Test Case 1: Enter a valid username with length 10 and alphanumeric characters.
Test Case 2: Enter a username with length less than 5 characters.
```

5.2 Web application that requires a password

Design and develop a login form for a web application that requires a Password using Equivalence class Testing.

Hint

Test cases:

```
Equivalence classes for Password:

Class 1: Valid passwords (length 6-20 characters, alphanumeric)
Class 2: Invalid passwords (length < 6)

For the Password:

Test Case 3: Enter a valid password with length 12 and alphanumeric characters.
Test Case 4: Enter a password with length less than 6 characters.
```

5.3 Students who have scored more than 60 in any one subject or all subjects.

Write and test a program to select the number of students who have scored more than 60 in any one subject or all subjects.

Hint

Program:

```
for (-----)
{
```

```
for (-----)
{
if(----)
{
String b= new String();
b=s.getCell(3,i).getContents();
```

Try:

1. Create a test suite to ensure that the bug fix doesn't introduce new issues, thus performing regression testing.

Hint

```
// White-box tests
void test Bubble Sort() {
    int arr1[] = {5, 1, 4, 2, 8};
    int arr2[] = {3, 5, 1, 9, 2};
    int arr3[] = {12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int arr4[] = {1, 2, 3, 4, 5};
    int arr5[] = {5, 4, 3, 2, 1};
```

6.Testing Technique: Decision Table Approach

The Decision Table Approach is a testing technique used to test systems that involve complex business rules or conditions. It helps in designing test cases to cover all possible combinations of inputs and conditions. Here are a few examples to illustrate how the Decision Table Approach works.

6.1 Discount based on their age and membership status.

Design test cases to ensure comprehensive coverage of all possible scenarios and conditions for a person is eligible for a discount based on their age and membership status.

Hint

Condition	Member (M)	Age>60 (A)	Discount (%)
Condition 1: Eligible for 20% discount	Yes	Yes	20
Condition 2: Eligible for 10% discount	Yes	No	10

Test Cases:

```
Test Case 1: Member (M) = Yes, Age > 60 (A) = Yes

Expected Result: Discount (%) = 20
Test Case 2: Member (M) = Yes, Age > 60 (A) = No
Expected Result: Discount (%) = 10
```

6.2. Loan Approval System:

Design test cases to ensure comprehensive coverage of all possible scenarios and conditions for Loan Approval System.

Hint

Suppose you're testing a loan approval system that determines whether an applicant is eligible for a loan based on their income and credit score. The decision table might look like this:

Income (Input A)	Credit Score (Input B)	Eligibility (Output)
Low	Low	Not Eligible
Low	High	Not Eligible

6.3 Traffic Light Control System:

Design test cases to ensure comprehensive coverage of all possible scenarios and conditions for Traffic Light Control System.

Hint

Imagine you're testing a traffic light control system that decides the signal pattern based on the time of day and traffic density. The decision table might look like this:

Time of Day (Input A)	Traffic Density (Input B)	Signal Pattern (Output)
Morning	Low	Green
Morning	High	Red

Try:

1. Experiment with different levels of code coverage (statement coverage, branch coverage, and path coverage) using a sample program. Analyse the effectiveness of each level in uncovering defects.

7.Testing Technique: Dataflow Testing

Dataflow testing is a white-box testing technique that focuses on the flow of data within a software application. It aims to identify potential errors in the processing of data as it moves through the system. Here are some examples to illustrate how dataflow testing works:

7.1 Area of A Rectangle Problem:

Design and develop a program that calculates the area of a rectangle. The program takes input for length and width, calculates the area, and then outputs the result.

Hint

```
length = input("Enter length: ")
width = input("Enter width: ")
```

Test Cases:

```
Valid Input Test Case:

Input: length = 5, width = 4
Expected Output: area = 20
```

7.2 Students' Grades Based on their Scores:

Design and develop a program that categorizes students' grades based on their scores.

Hint

Program:

```
score = input("Enter score: ")
grade = ""
if score >= 90:
grade = "A"
elif score >= 80:
```

Test Cases:

```
Input: score = 95
Expected Output: grade = "A"
Input: score = 85
Expected Output: grade = "B"
```

7.3 Factorial of A Given Number:

Design and develop a program that calculates the factorial of a given number and write down its test cases.

Hint

Program:

```
 while ----> 0:
 factorial *= num
 num -= 1
```

Test Cases:

```
Valid Input Test Cases:

Input: num = 5
Expected Output: factorial = 120 (5! = 5 * 4 * 3 * 2 * 1)
```

Try:

1. Experiment with different defect tracking and management tools to analyze their features, usability, and effectiveness in facilitating communication between testers and developers.

8. Testing Technique: Basis paths

8.1 Linear Search

Design, develop, code and run the program in any suitable programming language to implement the Linear search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and write the test results.

Hint

Program:

```
if(----)
{
printf("enter the elements in ascending order\n"); for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("enter the key element to be searched\n"); scanf("%d",&key);
succ=binsrc(a,0,n-1,key);
if(---)
printf("Element found in position = %d\n",succ+1); else
printf("Element not found \n");
}
else
printf("Number of element should be greater than zero\n"); return 0;
}
```

8.2 Binary Search

Design, develop, code and run the program in any suitable programming language to implement the binary search algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and write the test results.

Hint

Program:

```
int linearSearch(-----)
{
    for (int i = 0; i < n; i++)
{
        if (-----)
{
            return i; // Return the index of the element if found
        }
    }
}
```

8.3 Merge Sort

Design, develop, code and run the program in any suitable programming language to implement the merge sort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and write the test results.

Hint

Program:

```
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(------);
```

```
    printf("\nSorted array is \n");
    printArray(------);
    return 0;
}
```

Try:

1. Experiment with different levels of code coverage (statement coverage, branch coverage, and path coverage) using a sample program. Analyze the effectiveness of each level in uncovering defects.
2. Develop a simple Java class representing a stack and write unit tests using JUnit.

9. Testing Technique: Path Testing

9.1 Quick Sort

Design, develop, code and run the program in any suitable programming language to implement the Quicksort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and write the test results.

Hint

Program:

```
if(----)
{
printf("enter the elements of the array");
for(-----)
}
else
}
scanf("%d",&a[i]);
quicksort(a,0,n-1);
printf("the elements in the sorted array is:\n"); for(i=0;i<n;i++)
print f("%d\t",a[i]);
printf(“size of array is invalid\n”);
}
```

```
Path with no recursive calls: This path represents the case where the array is
already sorted or contains only one element, so no further recursive calls are
made. Test cases for this path should include arrays that are already sorted or
contain only one element.

Path with one recursive call: This path represents the case where one recursive
call is made to sort one of the sub-arrays. Test cases for this path should
include arrays with elements that require only one partitioning step.
```

9.2 Absolute Letter Grading

Design, develop, code and run the program in any suitable programming language to implement an absolute letter grading procedure, making suitable assumptions. Determine the basis paths and using them derive different cases, execute these test cases and write the test results.

Hint

Program:

```
        switch(----)
        {
        case 'a':printf("excellent\n");
        break;

        case 'b':printf("very good\n");
        break;
        case 'c':printf("good\n");
        break;
        case 'd':printf("above average\n");
        break;
        case 'e':printf("satisfactory\n");
        break;
        }
        printf("the percentage is %f and the grade is %c\n",per,grade);

        }
```

```
Path with Score in the A Range:

Input: Score = 95
Expected Output: Grade = A
Path with Score in the B Range:

Input: Score = 85
Expected Output: Grade = B
```

Try:

1. Investigate the challenges and best practices associated with testing in agile development environments. Design experiments to measure the impact of iterative development cycles on testing activities and defect discovery rates.

10. Black box Testing:

Black box testing involves testing a software application's functionality without knowing its internal code structure. Testers focus on input-output behaviour, specifications, and requirements. Here are five examples of black box testing methodologies:

10.1 State Transition Testing

Design and develop any suitable programming language for test a model the states and transitions of the vending machine (e.g., idle, selecting item, dispensing item) and design test cases to verify the correct behaviour of the machine during state transitions.

Hint

Program:

```
// Function to simulate selecting an item
void selectItem(enum VendingMachineState *currentState, float *funds, float
itemPrice) {
    if (-----)
```

```
{
        ------
    } else {
        printf("Cannot select item in current state.\n");
    }
}

// Function to simulate adding funds
void addFunds(enum VendingMachineState *currentState, float *funds, float amount)
{
    if (*currentState == SELECTING_ITEM) {
        *funds += amount;
        printf("Added $%.2f. Total funds: $%.2f\n", amount, *funds);
    } else {
        printf("Cannot add funds in current state.\n");
    }
}
```

```
States:

Idle: The vending machine is waiting for user input.
Selecting Item: The user is selecting an item to purchase.

Transitions:

Idle -> Selecting Item: Triggered when the user makes a selection.
Selecting Item -> Dispensing Item: Triggered when the user confirms the selection
and has sufficient funds.
```

Test Cases:

```
Test Case 1: User Selects Available Item:

Precondition: Vending machine is in the "Idle" state with available items.
Action: User selects an available item.
Expected Result: Vending machine transitions to the "Selecting Item" state and
prompts the user for payment.
Test Case 2: User Selects Unavailable Item:

Precondition: Vending machine is in the "Idle" state with some items out of stock.
Action: User selects an unavailable item.
Expected Result: Vending machine transitions to the "Out of Stock" state and
notifies the user that the item is unavailable.
```

10.2 Error Guessing

Design and develop any suitable programming language for test a file upload feature of a document management system. Use your intuition and past experience to identify potential error scenarios (e.g., invalid file format, exceeded file size limit) and design test cases to validate error handling.

Hint

Test Cases:

```
Test Case 1: Upload Invalid File Format:

Precondition: User is on the file upload page.
Action: User attempts to upload a file with an unsupported format (e.g., .exe).
```

```
Expected Result: System displays an error message indicating that the file format
is not supported.
Test Case 2: Upload Exceeded File Size Limit:

Precondition: User is on the file upload page.
Action: User attempts to upload a file that exceeds the maximum file size limit.
Expected Result: System displays an error message indicating that the file size
exceeds the maximum allowed limit.
```

Try:

1. Test Case: Addition Operation
Input: Two positive integers (e.g., 5, 7)
Expected Output: The sum of the two integers (e.g., 12)

2. Test Case: Subtraction Operation
Input: Two positive integers (e.g., 10, 3)
Expected Output: The difference between the two integers (e.g., 7)

3. Test Case: Division Operation
Input: Two positive integers (e.g., 10, 2)
Expected Output: The quotient of the two integers (e.g., 5)

10.3 Equivalence Partitioning

Design and develop a program to test a login form where valid usernames fall into one class and invalid ones fall into another, ensuring that a representative from each class is tested and write down its test cases.

Hint

Program:

```
    char username[20];

    // Test cases for valid usernames
    printf("Testing valid usernames:\n");
    printf("-------------------------\n");
    strcpy(username, "user1");
    if (-----) {
        printf("Test Passed: Username '%s' is valid\n", username);
    } else {
        printf("Test Failed: Username '%s' is invalid\n", username);
    }

    strcpy(---);
    if (validateUsername(username)) {
        printf("Test Passed: Username '%s' is valid\n", username);
    } else {
        printf("Test Failed: Username '%s' is invalid\n", username);
    }

    // Test cases for invalid usernames
    printf("\nTesting invalid usernames:\n");
    printf("---------------------------\n");
    strcpy(username, "invaliduser");
    if (!validateUsername(username)) {
        printf("Test Passed: Username '%s' is invalid\n", username);
```

```
    } else {
        printf("Test Failed: Username '%s' is valid\n", username);
    }
```

Test Cases:

```
Test Cases for Valid Usernames:

Valid Username Test 1:
Input: Username: "user1"
Expected Output: Login successful


Test Cases for Invalid Usernames:

Invalid Username Test 2:
Input: Username: "invaliduser"
Expected Output: Login failed
```

Try:

1. Test Case: Multiplication Operation

Input: Two positive integers (e.g., 4, 6)

Expected Output: The product of the two integers (e.g., 24)

2. Test Case: Square Root Operation
Input: A positive integer (e.g., 25)
Expected Output: The square root of the integer (e.g., 5)

11. White box Testing:

In software development, white box testing is a method where the internal structure, design, and implementation of the software are known to the tester. This allows for more detailed and comprehensive testing, as the tester can examine the code and assess its behaviour.

11.1 Statement Coverage:

Design and develop a program for test a function that sorts an array of integers using the bubble sort algorithm and Design test cases to achieve statement coverage, ensuring that each statement in the function is executed at least once.

Hint

Program:

```
for (-----) {
        for (-----) {
            if (arr[j] > arr[j+1]) {
                // Swap -----
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
```

Test Cases:

```
Test Case 1: Empty Array

Input: Empty array
Expected Output: No statements executed (edge case)
Test Case 2: Array with One Element

Input: [5]
Expected Output: Only one statement executed in the outer loop (edge case)
```

11.2 Branch Coverage:

Design and develop a program for test a function that calculates the factorial of a number and Design test cases to achieve branch coverage, ensuring that all decision points in the function (e.g., if conditions, loops) are exercised.

Hint

Program:

```
if (----) {
        return 0; // Invalid input, factorial of negative number is undefined
    }
    else if (n == 0) {
        return 1; // Base case: factorial of 0 is 1
    }
    else {
        unsigned long long fact = 1;
        for (int i = 1; i <= n; ++i) {
            fact *= i; // Calculate factorial using iterative method
        }
        return fact;
```

Test Cases:

```
Test Case 1: Factorial of Negative Number

Input: -5
Expected Output: 0 (executes the first if condition)
Test Case 2: Factorial of Zero

Input: 0
Expected Output: 1 (executes the second if condition)
```

11.3 Loop Testing

Design and develop a program for test a function that computes the sum of elements in an array using a loop and Design test cases to test different loop conditions (e.g., empty array, single-element array, multiple-element array) and verify the correctness of the loop logic.

Hint

Program:

```
int sumArray(-----)
{
```

```
    int sum = 0;
    for (----)
{
        sum += arr[i];
    }
}
```

Test Cases:

```
Test Case 1: Empty Array

Input: [], size = 0
Expected Output: 0 (sum of elements in an empty array)
Test Case 2: Single-Element Array

Input: [5], size = 1
Expected Output: 5 (sum of a single element)
```

Try:

1. For a function that calculates the factorial of a number, ensure that each line of code within the function is executed at least once.
2. For an if-else statement, design test cases that cover both the true and false conditions to ensure that both branches of the code are executed.
3. For a function with nested loops and multiple conditions, design test cases that traverse each unique path through the code, ensuring that all possible combinations of conditions are tested.

12.Test Plan Document

12.1 Library Management System

Design and develop a program for Library Management System and apply testing on it and create a test plan document for that application.

Hint

```
Pass criteria: librarians could use this GUI to interface with the backend library
database without any difficulties
Result: pass
Database test
Pass criteria: Results of all basic and advanced operations are normal (refer to
section 4) Result: pass
Basic function test Add a student
Pass criteria:
•Each customer/student should have following attributes: Student ID/SSN (unique),
Name, Address and Phone number.
Result: pass
```

12.2 E-commerce application

Design and develop a program for E-commerce application and create a test plan document for that application.

Hint

```
The following are a few things to test:

Is it going to auto scroll?
If yes, at what interval will the image be refreshed?
When the user hovers over it, is it still going to scroll to the next one?
Can it be hovered on?
Can it be clicked on?
If yes, is it taking you to the right page and right deal?
Is it loading along with the rest of the page or loads last in comparison to the other elements on the page?
Can the rest of the content be viewed?
Does it render the same way in different browsers and different screen resolutions?
```

Try:

1.Take any real time application and apply testing on it and create a test plan document for that application.

13.Automated Testing Tools:

13.1 Selenium

Describe a real-world scenario where Selenium is utilized effectively in software testing methodologies.

Hint

```
E-commerce Website Testing with Selenium

Background:
A leading e-commerce company is planning to launch a new version of its website with enhanced features and a revamped user interface. The development team follows Agile methodology for software development, with frequent iterations and continuous integration. To ensure the quality of the new website, the QA team decides to employ Selenium for automated testing.

Testing Objectives:

Validate the functionality of critical features such as user registration, product search, add to cart, and checkout process.
Verify the compatibility of the website across different web browsers (Chrome, Firefox, Safari).
Ensure responsiveness and usability across various devices (desktops, tablets, mobile phones).
Perform regression testing to detect any unintended side effects of new code changes.
```

13.2 Selenium Web Driver

Explain how Selenium WebDriver interacts with web browsers to automate testing activities.

Hint

```
Initialization:

First, the Selenium WebDriver needs to be instantiated within the test script. This involves importing the necessary Selenium libraries into the script and creating a WebDriver instance for the desired browser.
Browser Initialization:

Once the WebDriver instance is created, the test script instructs Selenium to launch the desired web browser (e.g., Chrome, Firefox, Safari). This is achieved by specifying the browser type as a parameter when creating the WebDriver instance.
```

Try:

1. Explain the importance of Selenium in software testing methodologies.
2. Compare and contrast Selenium with other automated testing tools.

13.3 Appium

Explain the significance of Appium as a mobile automation testing tool in modern software development practices.

Hint

```
Cross-Platform Compatibility: One of the most significant advantages of Appium is its cross-platform compatibility. With Appium, testers can write automation scripts once and execute them across both iOS and Android platforms. This cross-platform support streamlines the testing process, reduces development effort, and ensures consistent test coverage across different operating systems, thereby optimizing resource utilization and accelerating time-to-market for mobile applications.
```

Try:

1. What is JMeter, and how is it used in software testing?
2. What are the key features and components of JMeter?
3. How does JMeter facilitate load testing and performance testing?
4. Can you provide an example of how Bugzilla is used to track and manage bugs in a real-world software development project?

14. Test Management Tools

14.1 Test link

Describe the process of creating and managing test plans and test cases in Test Link.

Hint

```
Test Case Organization:
```

```
Test Link allows testers to create test suites, test plans, and test cases in a structured hierarchy.
Test suites help organize test cases based on functional areas or modules, while test plans define the scope, objectives, and schedule of testing activities.
Test Case Creation and Documentation:

Test Link provides a user-friendly interface for creating and documenting test cases.
Test cases can include detailed descriptions, preconditions, test steps, expected results, and other relevant information.
Testers can attach files, screenshots, or links to external resources to enhance the documentation of test cases.
```

14.2 TestRail

Describe the significance of TestRail as a test management tool in modern software development practices.

Hint

```
Test Case Organization and Reusability: TestRail enables systematic organization and categorization of test cases based on modules, features, or user stories. Test cases can be easily reused across multiple test runs or projects, saving time and effort associated with duplicating test cases. TestRail's hierarchical test suite structure allows teams to maintain a structured repository of reusable test assets, fostering consistency, scalability, and maintainability in testing efforts.
```

Try:

1. Can you explain the process of integrating Test Link with other testing tools or frameworks for improved test automation and reporting capabilities?
2. Explain how test plans are created and organized within Test Link to outline testing objectives, scope, and requirements.
3. Illustrate how Test Link facilitates version control and traceability of test cases, allowing for easy tracking of changes and updates over time.

15. Bug Tracking Tools

15.1 Bugzilla

Explain in detail how Bugzilla can be utilized as a testing tool, including its integration with testing frameworks, management of test cases, traceability features, and reporting capabilities.

Hint

```
Bugzilla, renowned primarily as a bug tracking system, indeed transcends its core functionality to become an indispensable asset in the realm of software testing. Despite not being conventionally labeled as a testing tool, Bugzilla's versatility and robust feature set make it an ideal companion throughout the testing process, seamlessly integrating with testing frameworks, facilitating test case management, ensuring traceability, and offering insightful reporting capabilities.
```

15.2 Jira

Describe how Jira's functionality supports testing efforts throughout the software development lifecycle.

Hint

```
Jira facilitates meticulous test planning by enabling teams to define testing
requirements, allocate resources, and outline test strategies comprehensively.
Through customizable project workflows and issue types, teams can create dedicated
testing tasks, epics, or user stories within Jira, encapsulating crucial details
such as testing objectives, scope, and dependencies. For instance, a QA team can
create a Jira epic titled "Regression Testing for Release X" and delineate
corresponding testing tasks encompassing functional, performance, and security
testing aspects. By leveraging Jira's intuitive interface and collaborative
features, teams can ensure alignment and clarity in test planning efforts.
```

Try:

1. Provide an example of how Bug Host is used to track and manage bugs in a real-world software development project?

16.Final Notes

Dealing with the raw data which has no longer been accessible and cannot be utilized should beprocessed to use efficiently. Then here comes data-wrangling which helps to turn non-resourceful (raw) data into valuable data which in turn returns valuable information. Each step in the process of wrangling the data is to the best possible analysis. The outcome is expected to generate a robust and reliable analysis of the data.

The problems in this tutorial are certainly NOT challenging. Check out these sites:

- The Springboard- (https://www.springboard.com/blog/data-science/software-testing/)
- Educative - https://www.educative.io/courses/ software-testing -with-python/coding-challenges-on-standardization
- iMocha - https://www.imocha.io/tests/ software-testing -with-python-test
- Kaggle - https://www.kaggle.com/code/prakharrathi25/ software-testing
- Github - https://moderndive.github.io/moderndive_labs/static/PS/PS03_ software-testing.html
- Rpubs - https://rpubs.com/Rokshana_ software-testing /891691
- Towardsdatascience - https://towardsdatascience.com/conquer-the-python-coding-round-in- software-testing -interviews-5e27c4513be3
- Into the minds - https://www.intotheminds.com/blog/en/the-11-challenges-of-data-preparation-and- software-testing /
- Posit - https://posit.co/blog/ software-testing -unruly-data/

Student must have any one of the following certification:

1. Coursera – Foundation for software testing and validation
2. Coursera – Software testing and automation
3. Coursera – Introduction to the software testing
4. Coursera – Introduction to Test and Behavior driven development
5. Geeks for Geeks – software-testing-basics
6. Stanford – Software Testing Strategies
7. Udemy – Automated Software Testing with Python
8. Pluralsight – Unit testing Legacy Code in Java SE
9. Harvard University - *software testing* and test-driven development.
10. Udacity – A/B Testing Fundamentals

V. TEXT BOOKS:

1. Kshira Sagar Naik Priyadarshini Tripathy, “Software Testing and Quality Assurance-Theory and Practice”, John Wiley and Sons Inc, Wiley Student Edition, 2010.
2. Jeff Tian, “Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement”, John Wiley and Sons, Inc., Hoboken, New Jersey, 2005.

VI. REFERENCE BOOKS:

1. Daniel Galin, “Software Quality Assurance - From Theory to Implementation”, Pearson Education Ltd UK, 2004.
2. Milind Limaye, “Software Quality Assurance”, TMH, New Delhi, 2011.

VII. ELECTRONICS RESOURCES

1. https://www.cisco.com/application/pdf/en/us/guest/products/ps2011/c2001/ccmigration 09186a00802342c f.pdfhttps://www.jntubook.com
2. http://ftp.utcluj.ro/pub/users/cemil/dwdm/dwdm/Intro/05311707.pdfhttps://archive.ics.uci.edu/ml/datasets.php

VIII. MATERIALS ONLINE

1. Course Template
2. Lab Manual