Practical 1

Analyse the given system and prepare functional and non-functional requirements.

Introduction

An expense management system is a software tool designed to streamline and automate the process of tracking, categorizing, and reporting on business expenses. It provides a centralized platform for employees to input and submit expense claims, while managers can review, approve, and reimburse them efficiently. By automating the expense management process, businesses can reduce manual errors, improve financial accuracy, and enhance overall operational efficiency.

Functional Requirement

1. User Authentication and Profile Management:
   - ➢ Users should be able to securely log in, manage their profiles, and reset passwords if needed.
2. Intuitive Expense Submission:
   - ➢ User-friendly interfaces for easily submitting expenses with clear fields for date, category, description, and amount.
   - ➢ Mobile accessibility for submitting expenses on-the-go.
3. Receipt Management:
   - ➢ Ability to upload receipts, with a user-friendly process for attaching receipts to expense entries.
   - ➢ Option for OCR to simplify receipt information entry.
4. Expense Tracking and Status:
   - ➢ Real-time tracking of submitted expenses and their approval/reimbursement status.
   - ➢ Notifications for submitted expenses and changes in status.
5. Mobile Accessibility:
   - ➢ Mobile apps or responsive design for easy access and submission from smartphones or tablets.
6. Approval Workflow Management:
   - ➢ Configurable approval workflows with multiple levels and conditions.
   - ➢ Notifications for pending approvals and the ability to delegate approval authority.
7. Policy Configuration and Compliance:
   - ➢ Configurable policy rules for different types of expenses.
   - ➢ Monitoring tools to ensure compliance with company policies.

8. Reporting and Analytics:

- ➢ Comprehensive reporting tools with customizable reports.
- ➢ Analytics for expense trends, budget adherence, and policy violations.

9. Reimbursement Processing:

- ➢ Automated calculation of reimbursable amounts.
- ➢ Tools for tracking and managing the reimbursement process.

Non-Functional Requirement

1. Scalability:
   - ➢ The system should scale seamlessly to accommodate a growing number of users and transactions.

2. Reliability:
   - ➢ High availability: The system should be available with minimal downtime.
   - ➢ Fault tolerance: Ability to continue functioning in the presence of errors or failures.

3. Security:
   - ➢ Data encryption: Sensitive data should be encrypted during transmission and storage.
   - ➢ Access control: Only authorized personnel should have access to specific functionalities.

4. Usability:
   - ➢ Intuitive user interface: The system should be user-friendly and easy to navigate.
   - ➢ Accessibility: Accessibility features for users with disabilities.

5. Maintainability:
   - ➢ Ease of updates: The system should allow for easy updates and modifications.
   - ➢ Documentation: Comprehensive documentation for administrators and users.

6. Compatibility:
   - ➢ Integration compatibility: Compatibility with various devices and third-party systems.

7. Auditability:
   - ➢ Detailed audit trail: Logging and tracking of user activities for auditing purposes.

8. Data Backup and Recovery:
   - ➢ Regular and secure data backups.
   - ➢ Efficient data recovery mechanisms in case of system failures.

9. Compliance:
   - ➢ Adherence to legal and regulatory requirements related to data protection and financial transactions.

10. Resource Utilization:
   - ➢ Efficient use of system resources such as memory and processing power.

11. Interoperability:
   - ➢ Integrate with existing accounting and ERP systems.
   - ➢ Open APIs for custom integrations.

Practical 2

Analyse the given system and prepare User story and Acceptance criteria.

User stories

User story-1 (Expense Submission):

As an employee, I want to easily submit expense reports on the go using the mobile app or web interface so that I can efficiently track my costs.

User story-2 (Receipt Upload):

As an employee, I want to effortlessly attach receipts to my expense reports.

User story-3 (Generate Report):

As an employee, I want to be able to generate reports on my expenses so that I can track my spending, identify trends, and make informed financial decisions.

Acceptance criteria

Functionality - 1	**Expense Submission**
Story - 1	As an employee, I want to easily submit expense reports on the go using the mobile app or web interface so that I can efficiently track my costs.
Scenario – 1.1	Users submit expense details based on the correct date, amount.
Acceptance Criteria – 1.1	**Given:** the user is on the submit expense page. **When:** the user selects date, expense category and enters amount, description (if required) and then press save button. **Then:** the user will be navigated to the dashboard and system generates a unique expense report ID and displays it to the user with success message: "Expense submitted".

Functionality - 1	**Expense Submission**
Story – 1	As an employee, I want to easily submit expense reports on the go using the mobile app or web interface so that I can efficiently track my costs.
Scenario – 1.2	Users submit expense details based on the incorrect date, amount.
Acceptance Criteria – 1.2	**Given:** the user is on the submit expense page. **When:** the user selects future date, enter incorrect value (negative or zero) for amount. **Then:** disable future date selection for expense date. Show popup message for the wrong amount.

Functionality – 1	**Expense Submission**
Story – 1	As an employee, I want to easily submit expense reports on the go using the mobile app or web interface so that I can efficiently track my costs.
Scenario – 1.3	Users submit expense details based on the without date, category and amount.
Acceptance Criteria – 1.3	**Given:** the user is on the submit expense page. **When:** the user clicks on "save" button. **Then:** show error message "Select Date". And show error message "Enter Amount". And show error message "Select Expense Category".

Functionality – 2	**Receipt Upload**
Story – 2	As an employee, I want to effortlessly attach receipts to my expense reports.
Scenario – 2.1	Users uploads receipt by selecting correct image/PDF of the receipt.
Acceptance Criteria – 2.1	**Given:** a valid receipt file (image or PDF) is selected for upload for an expense/expenses. **When:** the user clicks on the "Save" button. **Then:** the user will be navigated to the dashboard with the message: "Receipt has been uploaded".

Functionality - 2	**Receipt Upload**
Story – 2	As an employee, I want to effortlessly attach receipts to my expense reports.
Scenario – 2.2	Users uploads receipt by selecting incorrect file format (txt, docx, mp3, mp4)
Acceptance Criteria – 2.2	**Given:** the user is on the attach/update receipt page. **When:** the user selects a file that is not an image or PDF. **Then:** the system should display an error message stating "Invalid file format. Please upload an image or PDF file."

Functionality – 2	**Receipt Upload**
Story – 2	As an employee, I want to effortlessly attach receipts to my expense reports.
Scenario – 2.3	User attempts to upload receipt without selecting the receipt file.
Acceptance Criteria – 2.3	**Given:** the user is on the attach/update receipt page. **When:** the user clicks on the "Save" button without selecting the receipt file. **Then:** the system should display an error message stating "Please upload a receipt file to proceed."

Functionality – 3	**Generate Report**
Story - 3	As an employee, I want to be able to generate reports on my expenses so that I can track my spending, identify trends, and make informed financial decisions.
Scenario – 3.1	Users generates report based on the correct start date, end date, report type/format.
Acceptance Criteria – 3.1	**Given:** the user is in the reports section, **When:** the user selects past start date and end date, PDF/XLSX/CSV format and clicks on the “Export” button. **Then:** the user will be navigated to the dashboard. Display message “Report downloaded”.

Functionality – 3	**Generate Report**
Story – 3	As an employee, I want to be able to generate reports on my expenses so that I can track my spending, identify trends, and make informed financial decisions.
Scenario – 3.2	Users generates report based on the incorrect start date, end date, report type/format.
Acceptance Criteria – 3.2	**Given:** the user is in the 'Reports' section, **When:** the user selects future start date and/or end date and clicks on the “Export” button. **Then:** disable future date selection for start date and end date.

Functionality – 3	**Generate Report**
Story – 3	As an employee, I want to be able to generate reports on my expenses so that I can track my spending, identify trends, and make informed financial decisions.
Scenario – 3.3	Users generates report without start date, end date and report type/format.
Acceptance Criteria – 3.3	**Given**: the user is in the 'Reports' section. **When:** the user leaves required fields empty and clicks on the “Export” button. **Then:** show error message “Select start date”. And show error message “Select end date”. And show error message “Select report type/format”.

Practical 3

Write a Test case and test scenario.

Project Name:	**Expense Management System**	**Test Designed by:**	**Akshay D. Boricha**
Module Name:	**Expense Submission**	**Test Designed date:**	06-07-24
Release Version:	**1.0**	**Test Executed by:**	
		Test Execution date:	

Pre-condition	User should be on the submit expense page.

Test Case ID	Test Title	Test Type	Description
TC_001	Expense submission with valid inputs	Functional	Users submit expense details based on the correct date and amount.
TC_002	Expense submission with invalid inputs	Functional	Users submit expense details based on the incorrect date and amount.
TC_003	Verify expense submission page elements	GUI	Verify that all elements are available on expense submission page.

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Expense Submission	Test Designed date:	07-01-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_001	Test Execution date:	

Test Case Title	Expense submission with valid inputs
Test Type	Functional
Test Priority	High
Pre-condition	User should be on the submit expense page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select valid expense date in Date field	By clicking on date picker, calendar should popup to select past or current date		Pass		15-08-2023	
2	Enter valid amount in Amount field	Amount field should be editable and accept the positive amount.		Pass		1, 2, 3, ...	
3	Select Expense Category from dropdown	Drop down menu should display list of room type and user should be able to select any one from given list.		pass		Travelling, meals, entertainment, recharge, shopping, other	
4	Enter description in Description field (optional)	Description field should editable and accept the text input.		Pass		Rajkot to Ahmedabad travelling via bus	
5	Click on Save button	User should be navigated to the dashboard and system generates a unique expense report ID and displays it to the user with success message: "Expense submitted"		pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Expense Submission	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_002	Test Execution date:	

Test Case Title	Expense submission with invalid inputs
Test Type	Functional
Test Priority	Medium
Pre-condition	User should be on the submit expense page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select date of future days for expense date	Future date should be disabled		Pass		15-08-2030	
2	Enter negative values for expense amount	Negative values should not be entered in the text box.		Pass		0, -1, -2, -3, ...	
3	Verify that the user is not able to submit expense with blank expense date, expense amount, expense category.	Show error message "Select Date", "Enter Amount" and "Select Expense Category".		Pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Expense Submission	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_003	Test Execution date:	

Test Case Title	Verify expense submission page elements
Test Type	GUI
Test Priority	Medium
Pre-condition	User should be on the submit expense page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Verify that the expense submission page contains elements such as expense date, amount, category and description.	All listed control displayed properly on the page		Pass			
2	Verify that cursor is focused on Select expense date field on the page load.	Cursor is focused on select expense date.		Pass			
3	Verify that tab functionality is working properly or not	When tab pressed cursor move in next control.		Pass			
4	Verify that expense date has a valid placeholder of DD/MM/YYYY, Amount has placeholder of "Enter expense amount", Description has placeholder of "Enter expense description".	All text fields have proper placeholder.		Pass			
5	Verify that date picker option opens calendar of current month and year with proper navigation for past date selection.	Date picker opens successfully with current month and year and can be navigated to past dates.		Pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Receipt Upload	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
		Test Execution date:	

Pre-condition	User should on the attach receipt page.

Test Case ID	Test Title	Test Type	Description
TC_001	Receipt upload with valid input	Functional	Users upload receipt in the correct receipt file format.
TC_002	Receipt upload with invalid input	Functional	Users upload receipt in the incorrect receipt file format.
TC_003	Verify receipt upload page elements	GUI	Verify that all elements are available on receipt upload page.

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Receipt Upload	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_001	Test Execution date:	

Test Case Title	Receipt upload with valid input
Test Type	Functional
Test Priority	High
Pre-condition	User should be on the receipt upload page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select valid receipt file in receipt field	By clicking on upload receipt field, list of files and folders will be displayed. Select a receipt file in PNG, JPG, JPEG or PDF format. The file name will be displayed on the receipt field and the field should be editable.		Pass		bus_ticket.pdf	
2	Click on Save button	User should be navigated to the dashboard with the message: "Receipt has been uploaded".		pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Receipt Upload	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_002	Test Execution date:	

Test Case Title	Receipt upload with invalid input
Test Type	Functional
Test Priority	High
Pre-condition	User should be on the receipt upload page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select receipt file in invalid format	Show message with list of valid file formats: "Accepted file formats: PNG, JPG, JPEG, PDF"		Pass		Bus_ticket.docx	
2	Click on Save button	Show error message "Invalid file format. Please upload an image or PDF file".		Pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Receipt Upload	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_003	Test Execution date:	

Test Case Title	Verify receipt upload page elements
Test Type	GUI
Test Priority	Medium
Pre-condition	User should be on the receipt upload page.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Verify that the expense submission page contains elements such as attach receipt field and save button.	All listed control displayed properly on the page		Pass			
2	Verify that cursor is focused on attach receipt field on the page load.	Cursor is focused on attach receipt.		Pass			
3	Verify that tab functionality is working properly or not.	When tab pressed cursor move in next control.		Pass			
4	Verify that attach receipt field has a valid placeholder: "No files chosen".	All fields have proper placeholder.		Pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Generate Report	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
		Test Execution date:	

Pre-condition	User should be in the reports section.

Test Case ID	Test Title	Test Type	Description
TC_001	Generate report with valid inputs	Functional	Users generates report based on the correct start date, end date, report type/format.
TC_002	Generate report with invalid inputs	Functional	Users generates report based on the incorrect start date, end date, report type/format.
TC_003	Verify generate report page elements	GUI	Verify that all elements are available on generate report page.

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Generate Report	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_001	Test Execution date:	

Test Case Title	Generate report with valid inputs
Test Type	Functional
Test Priority	High
Pre-condition	User should be in the reports section.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select valid start-date	User should be able to select past start-date.		Pass		01-08-2022	
2	Select valid end-date	User should be able to select past end-date but the end-date should be greater than start-date.				01-08-2023	
3	Select expense category from dropdown (Optional)	Dropdown should provide list of categories to select expense category.		pass			
4	Select a valid report type/format.	Select a report type from given options (PDF, XSLX, CSV).		pass			
5	Click on Export button	User should be navigated to the dashboard with success message: "Report downloaded".				All, Travelling, meals, entertainment, recharge, shopping, other	

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Generate Report	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_002	Test Execution date:	

Test Case Title	Generate report with invalid inputs
Test Type	Functional
Test Priority	High
Pre-condition	User should be in the generate report section.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Select future start-date	Future start-date selection should be disabled.		Pass		05-10-2050	
2	Select future end-date	Future end-date selection should be disabled.					
3	Verify that the user is not able to export report with blank start-date, end-date and report type/format.	Show error message "Select start-date", "Select end-date", "Select report type".		Pass			

Project Name:	Expense Management System	Test Designed by:	Akshay D. Boricha
Module Name:	Generate Report	Test Designed date:	06-07-24
Release Version:	1.0	Test Executed by:	
Test Case ID	TC_003	Test Execution date:	

Test Case Title	Verify generate report page elements
Test Type	GUI
Test Priority	Medium
Pre-condition	User should be in the reports section.

Test Step	Test Case Description	Expected Result	Actual Result	Status	Comment	Data	Bug ID
1	Verify that the generate report page contains elements such as start-date, end-date, expense category dropdown and report type.	All listed control displayed properly on the page		Pass			
2	Verify that cursor is focused on start-date field on the page load.	Cursor is focused on start-date.		Pass			
3	Verify that tab functionality is working properly or not.	When tab pressed cursor move in next control.		Pass			
4	Verify that start-date and end-date has a valid placeholder: "dd/mm/yyyy" and expense category has a valid placeholder: "All".	All fields have proper placeholder.					
5	Verify that date picker options opens calendar of current month and year with proper navigation for past date selection.	Date picker opens successfully with current month, year and can be navigated to past dates also.		Pass			

Practical 4

Implement Equivalence Partitions

Equivalence partitioning is based on the principle that if one test case in a specific category detects a defect, it is likely that other test cases within the same category will also reveal the same defect. It involves dividing the input space of a software system into classes or partitions, where each partition is expected to exhibit similar behavior.

4.1 Let's consider a simple login form that accepts a username and password. The requirements for the login form are as follows:

- The username field must be alphanumeric and have a length between 5 and 10 characters.
- The password field must be alphanumeric and have a length between 8 and 12 characters.

Based on these requirements, create equivalence partitions for each input field.

Solution

For username:

Valid Class: username length must be between 5 to 10 character.

Invalid Class: username length is less than 5 character.

Invalid Class: username length is greater than 10 character.

For password:

Valid Class: password length must be between 8 to 12 character.

Invalid Class: username length is less than 8 character.

Invalid Class: username length is greater than 12 character.

4.2 Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage. Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page.

- If percentage entered by user is less than 50 %or more than 90 %, that equivalence partitioning method will show an invalid percentage.
- If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage

Solution

Valid Class: percentage must be between 50-90.

Invalid Class 1: percentage value is less than 50.

Invalid Class 2: percentage value is greater than 90.

4.3 Let us consider an example of software application. There is function of software application that accepts only particular number of digits, not even greater or less than that particular number. Consider an OTP number that contains only 6-digit number, greater and even less than six digits will not be accepted, and the application will redirect customer or user to error page.

- If password entered by user is less or more than six characters, that equivalence partitioning method will show an invalid OTP.
- If password entered is exactly six characters, then equivalence partitioning method will show valid OTP.

Solution

Valid Class: number length must be 6.

Invalid Class 1: number length is less than 6.

Invalid Class 2: number length is greater than 6.

4.4 Testing an input box for a mobile number accepting ten digits (i.e. length of input value has to be ten).

Solution

Valid Class: number length must be 10.

Invalid Class 1: number length is less than 10.

Invalid Class 2: number length is greater than 10.

4.5 Consider this scenario: For a bank savings account, a 4% interest rate is given if the account balance is between $01 and $200, 7% rate of interest is given if the account balance is in the range of $201 to 900, and 9% interest rate is considered for account balance is $901 & above. For the above scenario, we can identify three valid equivalence classes and one invalid.

Solution

Test cases for the above (interest rate) scenario:

Invalid data class: balance less than $01

Valid data class 1: balance between $1 and $200

Valid data class 2: balance between $201 and $900

Valid data class 3: balance greater than $900

Practical 5

Implement Boundary-Value Analysis

5.1 Let's consider a system that accepts Age 18 – 56. The requirements for the system are as follows:

- Minimum boundary value is 18
- Maximum boundary value is 56
- Valid Inputs: 18,19,55,56 and Invalid Inputs: 17 and 57

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of the age values.

Solution:

BOUNDARY VALUE TEST CASE		
INVALID TEST CASE (Min Value – 1)	**VALID TEST CASES** (Min, +Min, Max, -Max)	**INVALID TEST CASE** (Max Value + 1)
17	18, 19, 55, 56	57

Test case 1: Enter the value 17 (18-1) = Invalid

Test case 2: Enter the value 18 = Valid

Test case 3: Enter the value 19 (18+1) = Valid

Test case 4: Enter the value 55 (56-1) = Valid

Test case 5: Enter the value 56 = Valid

Test case 6: Enter the value 57 (56+1) = Invalid

5.2 Let's consider a system that have to test a text field (Name) which accepts the length between 6-12 characters.

- Minimum boundary value is 6
- Maximum boundary value is 12
- Valid text length is 6, 7, 11, 12
- Invalid text length is 5, 13

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of the text length.

Solution:

BOUNDARY VALUE TEST CASE		
INVALID TEST CASE (Min Value – 1)	**VALID TEST CASES** (Min, +Min, Max, -Max)	**INVALID TEST CASE** (Max Value + 1)
5	6, 7, 11, 12	13

Test case 1: Enter the value 5 (6-1) = Invalid

Test case 2: Enter the value 6 = Valid

Test case 3: Enter the value 7 (6+1) = Valid

Test case 4: Enter the value 11 (12-1) = Valid

Test case 5: Enter the value 12 = Valid

Test case 6: Enter the value 13 (12+1) = Invalid

5.3 Let's consider a system that calculates the shipping cost based on the weight of a package. The requirements for the system are as follows:

- Weight must be a positive numeric value.
- The shipping cost is determined based on weight ranges: 0-10 kg, 11-20 kg, and 21-30 kg.
- The shipping cost for each weight range is $10, $15, and $20, respectively.

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of the weight ranges.

Solution

BOUNDARY VALUE TEST CASE		
INVALID TEST CASE (Min Value – 1)	**VALID TEST CASES** (Min, +Min, Max, -Max)	**INVALID TEST CASE** (Max Value + 1)
-1	0, 1, 9, 10	11
10	11, 12, 19, 20	21
20	21, 22, 29, 30	31

For shipping cast $10:

Test case 1: Enter the value -1 (0 -1) – Invalid

Test case 2: Enter the value 0 – valid

Test case 3: Enter the value 1 (0+1) – valid

Test case 4: Enter the value 9 (10-1) – valid

Test case 5: Enter the value 10 – valid

Test case 6: Enter the value 11 (10+1) – Invalid

For shipping cast $20:

Test case 1: Enter the value 10 (11 - 1) – Invalid

Test case 2: Enter the value 11 – valid

Test case 3: Enter the value 12 (11 + 1) – valid

Test case 4: Enter the value 19 (20 - 1) – valid

Test case 5: Enter the value 20 – valid

Test case 6: Enter the value 21 (20 + 1) – Invalid

For shipping cast $30:

Test case 1: Enter the value 20 (21 - 1) – Invalid

Test case 2: Enter the value 21 – valid

Test case 3: Enter the value 22 (21 + 1) – valid

Test case 4: Enter the value 29 (30 - 1) – valid

Test case 5: Enter the value 30 – valid

Test case 6: Enter the value 31 (30 + 1) – Invalid

5.4 Let us assume there is a test case that takes the car speed from 40 km/hr- 80 km/hr, and if the speed is lower than 40 and higher than 80, then our software should send the notification.

- If the values are entered below 40 and above 80 then it will be considered invalid case.
- If the value entered is between 40-80 then it will be a valid case.

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of car speed.

Solution

BOUNDARY VALUE TEST CASE		
INVALID TEST CASE (Min Value – 1)	**VALID TEST CASES** (Min, +Min, Max, -Max)	**INVALID TEST CASE** (Max Value + 1)
39	40, 41, 79, 80	81

Test case 1: Enter the value 39 (40 - 1) = Invalid

Test case 2: Enter the value 40 = Valid

Test case 3: Enter the value 40(40 + 1) = Valid

Test case 4: Enter the value 79 (81 - 1) = Valid

Test case 5: Enter the value 80 = Valid

Test case 6: Enter the value 81(81 + 1) = Invalid

5.5 Consider the Order Pizza Text Box's behavior. Pizza values ranging from 1 to 10 are deemed legitimate. The message "success" is shown. While values 11 to 99 are deemed invalid for ordering, an error notice stating "Only 10 Pizza may be ordered" will show.

- Any number submitted in the Order Pizza form that is larger than 10 (let's say 11) is deemed invalid.
- Any number that is less than 1 and is 0 or below is deemed invalid.
- The numbers 1 to 10 are accepted as acceptable.
- Any three-digit number, such as -100, is unusable.

Based on these requirements, apply boundary-value analysis to identify test cases at the boundaries of pizza quantity.

Solution

Test case 1: Enter the value -1 = Invalid

Test case 2: Enter the value 0 = Valid

Test case 3: Enter the value 1 (0 + 1) = Valid

Test case 4: Enter the value 9 (10 - 1) = Valid

Test case 5: Enter the value 10 = Valid

Test case 6: Enter the value 11 (10 + 1) = Invalid

Practical 6

Understanding Decision Table Testing

6.1 Let's consider a simple login screen of any website. So create a decision table for a login functionality.

- If the user provides the correct username and password the user will be redirected to the homepage.
- If any of the input is wrong, an error message will be displayed.

Based on these requirements, create a decision table to systematically test all possible combinations.

Solution

Decision table for login functionality

Conditions	User name	Password	Output/Outcome
Case 1	False	False	Error message
Case 2	True	False	Error message
Case 3	False	True	Error message
Case 4	True	True	Navigate to home page

Test Case 1 – Username and password both are wrong. The user is shown an error message.

Test Case 2 – Username was correct, but the password was wrong. The user is shown an error message.

Test Case 3 – Username was wrong, but the password was correct. The user is shown an error message.

Test Case 4 – Username and password both were correct, and the user navigated to the home Screen

6.2 Let's consider a dialogue box that will ask the user to upload a photo with certain conditions. The requirements for the photo upload are as follows:

- You can upload only '.jpg' format image
- file size less than 32kb
- resolution 137*177.
- If any of the conditions fails the system will throw a corresponding error message stating the issue and if all conditions are met photo will be updated successfully

Based on these requirements, create a decision table to systematically test all possible combinations.

Solution

Conditions	Format	Size	Resolution	Output
Case 1	.jpeg	Less than 50 kb	130*170	Image uploaded
Case 2	.jpeg	Less than 50 kb	Not 130*170	Error message resolution mismatch
Case 3	.jpeg	Less than 50 kb	130*170	Error message resolution mismatch
Case 4	.jpeg	Less than 50 kb	Not 130*170	Error message resolution mismatch
Case 5	Not .jpeg	Less than 50 kb	130*170	Error message for format mismatch
Case 6	Not .jpeg	Less than 50 kb	Not 130*170	Error message for format and resolution mismatch
Case 7	Not .jpeg	More than 50 kb	130*170	Error message for format and size mismatch
Case 8	Not .jpeg	More than 50 kb	Not 130*170	Error message for format, size and resolution mismatch

6.3 Let's consider a piece of software determining whether a student is eligible for a scholarship based on their grades and extracurricular activities. The requirements are as follows:

- If percentage is greater than 90% and extracurricular activities are greater than 2 than only scholarship will be given.
- If both the condition are true than and only than scholarship will be given, otherwise not given.

Based on these requirements, create a decision table to systematically test all possible combinations.

Solution

Conditions	Percentage >90%	Activities >2	Output
Case 1	True	True	Student is eligible for a scholarship
Case 2	False	True	Error message
Case 3	True	False	Error message
Case 4	False	False	Error message

Test Case 1 – Percentage and Activities both were correct, and the student is eligible for a scholarship.

Test Case 2 – Activities was correct, but the Percentage was wrong. The user is shown an error message.

Test Case 3 – Percentage was correct, but the Activities was wrong. The user is shown an error message.

Test Case 4 – Percentage and Activities both are wrong. The user is shown an error message.

6.4 Let's consider a simple e-commerce system that calculates discounts based on the customer type and the total order amount. The requirements for the discount calculation are as follows:

1. If the customer is a regular customer:
   - If the total order amount is less than $100, no discount is applied.
   - If the total order amount is between $100 and $500 (inclusive), a 10% discount is applied.
   - If the total order amount is greater than $500, a 15% discount is applied.
2. If the customer is a premium customer:
   - If the total order amount is less than $100, a 5% discount is applied.
   - If the total order amount is between $100 and $500 (inclusive), a 15% discount is applied.
   - If the total order amount is greater than $500, a 20% discount is applied.

Based on these requirements, create a decision table to systematically test all possible combinations

Solution

Decision table for item discount

Condition	Customer type	Order amount	Discount (%)
Case 1: regular, <$100	Regular	< $100	0
Case 2: regular, $100 - $500	Regular	$100 - $500	10
Case 3: regular, >$500	Regular	>$500	15
Case 4: premium, <$100	Premium	< $100	5
Case 5: premium, $100 - $500	Premium	$100 - $500	15
Case 6: premium, >$500	Premium	>$500	20

Test case 1: The customer is a regular member and the total amount is less than $100, the outcome will be a 0% discount on the total amount in the shopping cart.

Test case 2: The customer is a regular member and the total amount is between $100 - $500, the outcome will be a 10% discount on the total amount in the shopping cart.

Test case 3: The customer is a regular member and the total amount is greater than $500, the outcome will be a 15% discount on the total amount in the shopping cart.

Test case 4: The customer is a premium member and the total amount is less than $100, the outcome will be a 5% discount on the total amount in the shopping cart.

Test case 5: The customer is a premium member and the total amount is between $100 - $500, the outcome will be a 15% discount on the total amount in the shopping cart.

Test case 6: The customer is a premium member and the total amount is greater than $500, the outcome will be a 20% discount on the total amount in the shopping cart.

OR

Decision table for item discount

TestcaseID	Cust_type Regular	Cust_type Premium	<$100	$100 - $500	>$500	Discount %
Case 1	T	F	T	F	F	0
Case 2	T	F	F	T	F	10
Case 3	T	F	F	F	T	15
Case 4	F	T	T	F	F	5
Case 5	F	T	F	T	F	15
Case 6	F	T	F	F	T	20

Practical 7

Draw a control flow diagram and apply cyclomatic complexity for the given codes. Be sure about following points.

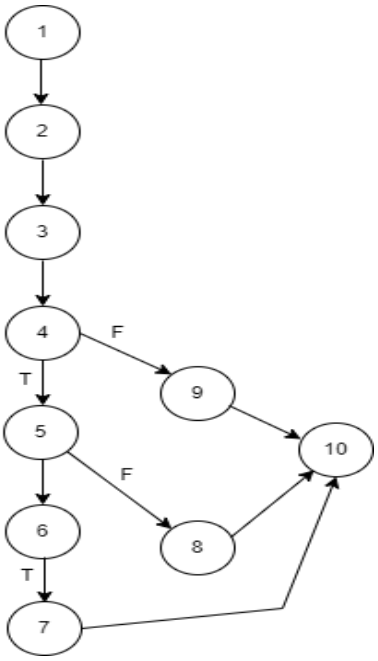
- Guarantees that all independent execution path is exercised at least once.
- Guarantees that both the true and false side of all logical decisions are exercised.
- Executes the loop at the boundary values and within the boundaries.
- Identify numbers of independence path require for testing.

Exercise 1:

```
void main(){
        int i, j, k;
        readln (i, j, k);
        if ( (i < j) || ( i > k) ){
                writeln (“then part”);
                if (j < k)
                        writeln (“j less then k”);
                else writeln (“j not less than k”);}
        else writeln (“else Part”);
}
```

Solution

Control flow graph:



Cyclomatic complexity:

N = No. of nodes = 10

E = No. of edges = 11

V(G) = E - N + 2 = 11 - 10 + 2 = 3

P=No. of predicate node= 3

V(G) = P+1= 2 + 1 = 3

Independent paths:

Path 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 10

Path 2: 1 – 2 – 3 – 4 – 5 – 6 – 8 – 10

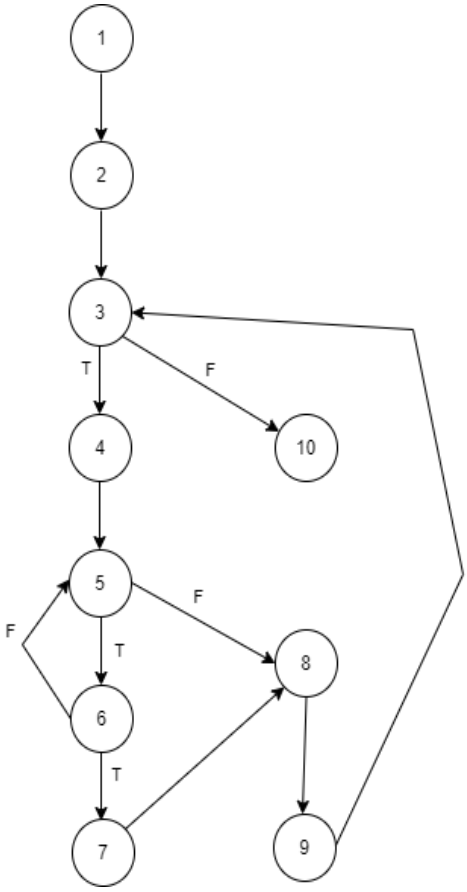
Path 3: 1 – 2 – 3 – 4 –9 – 10

Exercise 2:

```
i = 0;
n=4; //N-Number of nodes present in the graph
while (i<n-1) do
        j = i + 1;
        while (j<n) do
                if A[i]<A[j] then
                        swap(A[i], A[j]);
        end do;
        i=i+1;
end do;
```

Solution

Control flow graph:



Cyclomatic complexity:

N = No. of nodes = 10

E = No. of edges = 12

V(G) = E - N + 2 = 12 - 10 + 2 = 4

P=No. of predicate node = 4

V(G) = P+1= 3 + 1 = 4

Independent paths:

Path 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 3 – 10

Path 2: 1 – 2 – 3 – 4 – 5 – 6 – 5 – 8 – 9 – 3 – 10

Path 3: 1 – 2 – 3 – 4 – 5 – 8 – 9 – 3 – 10

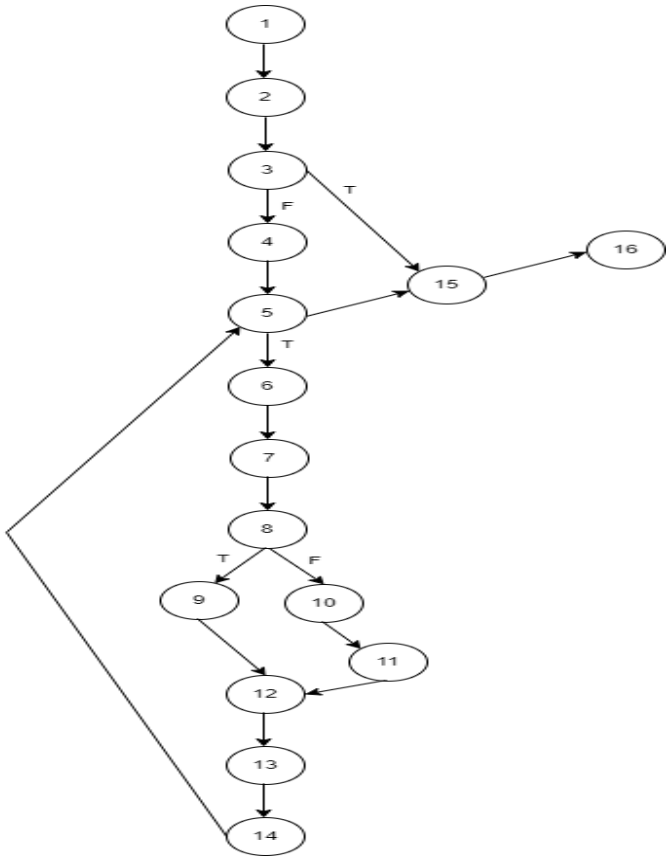
Path 4: 1 – 2 – 3 – 10

Exercise 3:

```
public Hashtable countAlphabet(String aString)
{
        Hashtable table = new Hashtable();
        If (aString.length > 4000)
                return table;
        StringBuffer buffer = new StringBuffer(aString);
        While (buffer.length() > 0){
                 String firstChar = buffer.substring(0, 1);
                Integer count = (Integer)table.get(firstChar);
                if (count == null){
                        count = new Integer(1);
                }
                else{
                         count = new Integer(count.intValue() + 1);
                }
        table.put(firstChar, count);
        buffer.delete(0, 1);
        }
        Return table;
}
```

Solution

Control flow graph:



Cyclomatic complexity:

N = No. of nodes = 16

E = No. of edges = 18

V(G) = E - N + 2 = 18 - 16 + 2 = 4

P=No. of predicate node = 4

V(G) = P+1= 3 + 1 = 4

Independent paths:

Path 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 12 – 13 – 14 – 5 – 15 – 16

Path 2: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – 13 – 14 – 5 – 15 – 16

Path 3: 1 – 2 – 3 – 4 – 5 – 15 – 16

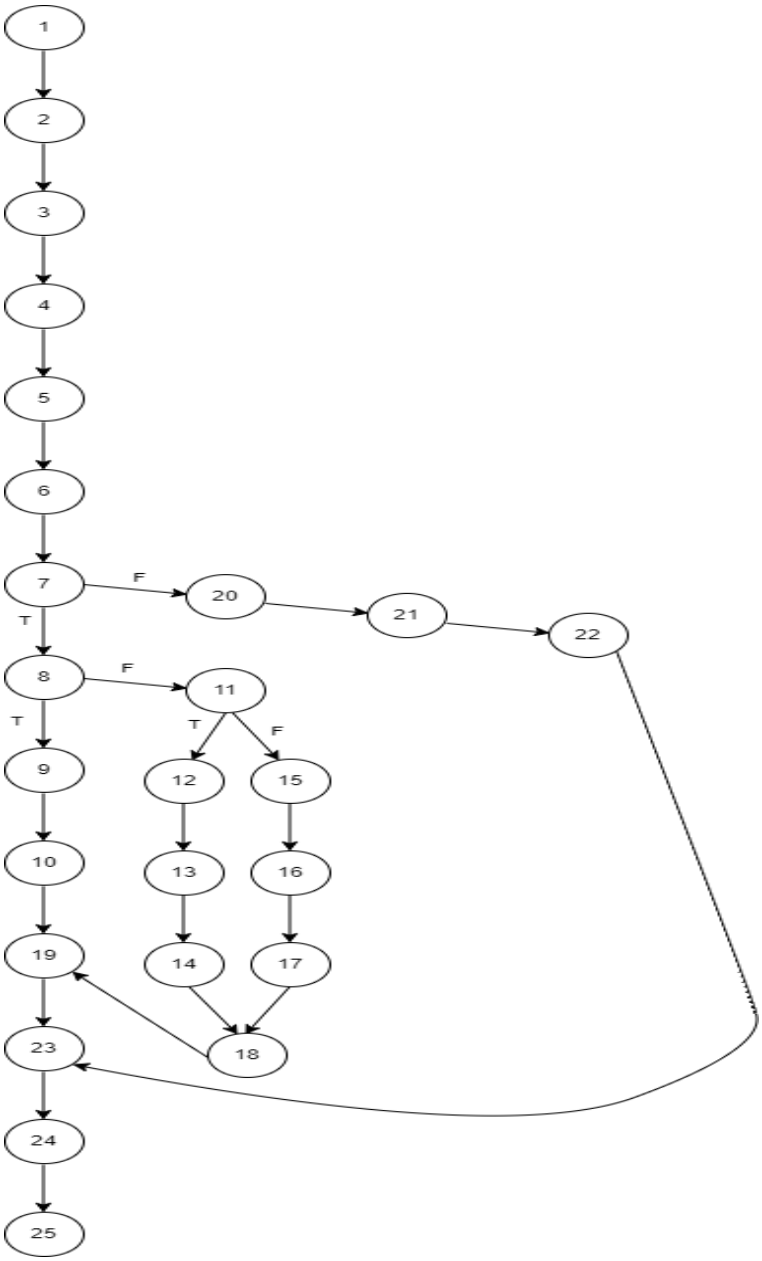
Path 4: 1 – 2 – 3 – 15 – 16

Exercise 4:

```
public class CyclomaticComplexityDemo {
public static void main(String[] args) {
// TODO Auto-generated method stub
        int var1 = 10;
        int var2 = 9;
        int var3 = 8;
        int var4 = 7;
        if (var1 == 10){
                if(var2 > var3){
                        var2 = var3;
                }
                else{
                        if (var3 > var4){
                                var3 = var4;
                        }
                        else{
                                var4 = var1;
                        }
                }
        }
        else{
        var1=var4;
        }
        System.out.println("Printing value for var1, var2, var3, and
var4"+var1+" "+var2+" "+var3+" "+var4);
        }
}
```

Solution

Control flow graph:



Cyclomatic complexity:

N = No. of nodes = 25

E = No. of edges = 27

V(G) = E - N + 2 = 27 - 25 + 2 = 4

P=No. of predicate node = 4

V(G) = P+1= 3 + 1 = 4

Independent paths:

Path 1: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 19 – 23 – 24 – 25

Path 2: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 11 – 12 – 13 – 14 – 18 – 19 – 23 – 24 – 25

Path 3: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 11 – 12 – 15 – 16 – 17 – 18 – 19 – 23 – 24 – 25

Path 4: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 20 – 21 – 22 – 23 – 24 – 25

Practical 8

To study and perform sample testing using Selenium Testing Tools. Answer following questions:

1. What is selenium? use of selenium?

Answer:

Selenium is an open-source tool for automating web browsers. It is primarily used for testing web applications, but it can also be used for web scraping, automating tasks, and other purposes. Selenium supports multiple programming languages, including Java, Python, Ruby, C#, and JavaScript.

Use of Selenium:

Here are some of the main uses of Selenium:

- **Automated Testing**: Selenium is widely used for automated testing of web applications. It allows testers to write test scripts that can interact with the application, verify its functionality, and report any errors.
- **Web Scraping**: Selenium can be used for web scraping, which involves extracting data from websites. It can be used to scrape data from websites that use a lot of JavaScript or have complex web pages.
- **Automating Tasks**: Selenium can be used to automate repetitive tasks, such as filling out forms, clicking buttons, and extracting data from websites.
- **Monitoring Website Performance**: Selenium can be used to monitor website performance, such as page load times, response times, and user experience

2. How it works?

Answer:

Selenium works by automating interactions with web browsers, allowing testers and developers to write scripts that simulate user behavior on websites. The tool can perform tasks like clicking buttons, filling out forms, navigating between pages, and verifying content.

Here's a simplified breakdown of how Selenium works:

1. **Choose a Programming Language:** Selenium supports multiple programming languages. You select one (e.g., Java, Python, C#) based on your preferences and project requirements.
2. **Write Test Scripts:** You write automated test scripts using the Selenium API in your chosen language. These scripts define the actions you want the browser to perform, such as navigating to a URL, clicking buttons, entering text, and verifying elements on the page.
3. **Load the Driver:** The test script loads the appropriate browser driver for the browser you want to control. This driver establishes a connection between your script and the browser instance.
4. **Initialize the Browser:** The script initializes the browser, specifying options like the browser's size, location, and any desired preferences.
5. **Execute Commands:** The script sends commands to the browser driver using the Selenium API. These commands instruct the browser to perform specific actions, such as navigating to a URL, finding elements on the page, and interacting with them.
6. **Verify Results:** After executing commands, the script verifies the results to ensure that the web application is behaving as expected. This might involve checking element visibility, text content, or other conditions.
7. **Close the Browser:** Once the test is complete, the script closes the browser instance.

3. Advantages and disadvantages of selenium.

Answer:

Advantages of Selenium:

1. **Open Source and Free**:
   - Selenium is free to use, which makes it a cost-effective solution for test automation. There are no licensing fees, making it accessible to individuals and companies of all sizes.
2. **Cross-Browser Support**:
   - Selenium supports multiple web browsers (Chrome, Firefox, Safari, Edge, etc.), enabling cross-browser testing. This ensures that web applications work correctly across different browsers.
3. **Multi-Language Support**:
   - Selenium supports multiple programming languages, including Java, Python, C#, Ruby, and JavaScript. Testers can write automation scripts in the language they are most comfortable with.
4. **Platform Independent**:
   - Tests can be run on different operating systems like Windows, macOS, and Linux, making it highly versatile for various environments.
5. **Active Community and Ecosystem**:
   - Selenium has a large and active user community, which provides a wealth of resources, tutorials, and support. There are also various integrations with third-party tools for CI/CD, reporting, and test management.
6. **Integrations with Testing Frameworks**:
   - Selenium can be integrated with popular test frameworks like TestNG, JUnit, NUnit, and others, allowing structured test case management and reporting.
7. **Scalability**:
   - Selenium can handle complex automation needs for large-scale projects. It can automate test cases for different scenarios, from simple functional tests to complex end-to-end testing.
8. **Flexibility for Automation**:
   - Selenium can automate various user actions like form submission, clicking, navigation, and even JavaScript interactions. This flexibility makes it suitable for a wide range of testing tasks.

Disadvantages of Selenium:

1. **Limited to Web Applications**:
   - Selenium can only automate web applications. It doesn't support desktop applications or mobile apps directly. For mobile testing, third-party tools like Appium are needed.

2. **Requires Programming Knowledge**:
   - Using Selenium requires knowledge of programming languages and test automation frameworks. Non-programmers might find it difficult to use without adequate training or experience.

3. **Difficult to Handle Dynamic Elements**:
   - Handling dynamic web elements (elements that change frequently, such as pop-ups, AJAX elements, etc.) can be challenging. Custom wait conditions (e.g., WebDriverWait) or complex locators are often needed.

4. **Manual Effort for Test Maintenance**:
   - Automated tests can become brittle when there are frequent changes to the web application's UI. This requires constant maintenance to update locators, scripts, and test logic, which can be time-consuming.

5. **No Built-in Support for Capturing Screenshots and Videos**:
   - While Selenium can capture screenshots during test execution, it lacks built-in support for video recording. Additional tools or libraries are required if video evidence is needed.

6. **Limited Support for Image-Based Testing**:
   - Selenium struggles with visual or image-based testing (i.e., verifying UI designs, colors, or layouts). For visual testing, tools like Applitools or Sikuli may be required.

7. **Cannot Handle CAPTCHA or OTP**:
   - Selenium cannot handle CAPTCHA, OTP (One-Time Password), or other security mechanisms directly, as these require manual intervention or third-party services.

8. **Requires External Tools for Test Management**:
   - Selenium lacks test case management features, meaning you need external tools to manage your test cases and test data effectively.

Practical 9

To study and perform Unit testing using JUnit. Answer following questions:

1. What is Junit? Explain use of Junit.

Answer:

JUnit is a **popular open-source testing framework** for Java programming that allows developers to write and run repeatable unit tests. It helps to ensure that individual units of source code (such as methods or classes) function as expected. JUnit is a crucial tool for **Test-Driven Development (TDD)** and for maintaining code quality, as it enables the automation of testing during the software development lifecycle.

Uses of JUnit:

1. **Unit Testing**:
   - JUnit is primarily used to write and run **unit tests**, which are small, isolated tests for individual units of code (usually methods or classes). This ensures that each component of your software behaves correctly.

2. **Test Automation**:
   - JUnit allows the **automation of testing** in the development pipeline. Automated tests can be executed as part of Continuous Integration (CI) pipelines to quickly identify bugs and regressions in the codebase.

3. **Test-Driven Development (TDD)**:
   - JUnit plays a crucial role in **Test-Driven Development**. Developers write tests first (before the code), ensuring that the code is written to pass those tests. JUnit helps verify that the code satisfies the test cases as development progresses.

4. **Regression Testing**:
   - As you modify or add new code, JUnit tests can be re-run to ensure that new changes do not break any existing functionality. This is called **regression testing**, which helps prevent the introduction of new bugs into the system.

5. **Integration Testing**:
   - Although JUnit is mainly used for unit testing, it can also be used for **integration testing**, where you test how different modules or components interact with each other.

6. **Mocking and Dependency Injection**:
   - JUnit can be used alongside mocking frameworks like **Mockito** to mock dependencies, making it easier to isolate the unit of code under test and ensure that external dependencies don't affect the test results.

2. Explain all methods of assert class of JUnit.

Answer:

Methods of Assert Class in JUnit:

1. **assertEquals(expected, actual):** Verifies that the expected value is equal to the actual value.

2. **assertNotEquals(unexpected, actual):** Verifies that the unexpected value is not equal to the actual value.

3. **assertSame(expected, actual):** Verifies that the expected object is the same as the actual object.

4. **assertNotSame(unexpected, actual):** Verifies that the unexpected object is not the same as the actual object.

5. **assertTrue(condition):** Verifies that the condition is true.

6. **assertFalse(condition):** Verifies that the condition is false.

7. **assertNull(actual):** Verifies that the actual object is null.

8. **assertNotNull(actual):** Verifies that the actual object is not null.

9. **assertArrayEquals(expected, actual):** Verifies that the expected array is equal to the actual array.

10. **assertIterableEquals(expected, actual):** Verifies that the expected iterable is equal to the actual iterable.

11. **fail(message):** Fails the test with the specified message.

3. Consider a simple Java class called **Calculator** that performs basic arithmetic operations. To perform unit testing using JUnit, you can create a test class and write test methods to verify the behaviour of the **addition() method**.

Solution

//class calculator

```
import java.util.Scanner;
public class calci {
  public static void main(String[] args) {
    char operator;
    int number1, number2;
    calci c=new calci(); // create an object of Scanner class
    Scanner input = new Scanner(System.in);

     System.out.println("Choose an operator: +, -, *, or /");      // ask users to enter operator
    operator = input.next().charAt(0);
    // ask users to enter numbers
    System.out.println("Enter first number");
    number1 = input.nextInt();
    System.out.println("Enter second number");
    number2 = input.nextInt();
    int x;
   if(operator=='+')
      x=c.addition(number1,number2);
  else if(operator=='-')
    x=c.subtraction(number1,number2);
  else if(operator=='*')
    x=c.multiplication(number1,number2);
  else
    x=c.division(number1,number2);
  System.out.println("answer is:"+x);
}
 int addition(int n1,int n2)
 {
  return n1+n2;
 }
 int subtraction(int n1,int n2)
```

```
 {
  return n1-n2;
 }
 int multiplication(int n1,int n2)
 {
  return n1*n2;
 }
 int division(int n1,int n2)
 {
  return n1/n2;
 }
}
```

//class test

```
import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;

public class test {
   calci c;

   @Before
   public void setUp() {
      c=new calci();
      }
   @Test
   public void testAdd_Positive() {
            int result = c.addition(5, 10);
      assertEquals(15, result);
   }
   @Test
   public void testAdd_PositiveNegative() {
      int result = c.addition(10, -5);
      assertEquals(4, result);
   }
   // Test case for addition of two negative numbers
   @Test
   public void testAdd_Negative() {
```

```
        int result = c.addition(-3, -7);
        assertEquals(-10, result);
    }
    // Test case for addition involving zero
    @Test
    public void testAdd_ZeroPositive() {
        int result = c.addition(0, 10);
        assertEquals(10, result);
    }
     /*Test case for addition of large numbers */
    @Test
    public void testAdd_LargeNumbers() {
        int result = c.addition(10000000, 2000000);
        assertEquals(12000000, result);
    }
}
```

4. Consider a Java class called **InterestCalculator** that calculates the interest amount based on a principal amount and an interest rate. To perform unit testing for the **InterestCalculator** class using JUnit, you can create a test class and write test methods to verify the behaviour of the **calculateInterest** method.

Solution

//class Interest calculator

```
import java.util.Scanner;

public class simple_interest {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.println("Enter the principal (amount), time, and rate::\n");
      float p = in.nextFloat(); // value of the principal
      float t = in.nextFloat(); // value of the time
      float r = in.nextFloat(); // value of the rate
      CalculateSimpleInterest(p, t, r);
   }
   // This method will calculate the simple interest
   public static void CalculateSimpleInterest(float x, float y, float z) {
      float SI; // Value of the simple interest
            SI = (x * y * z) / 100;
      System.out.println("\nSimple Interest = " + SI);
   }
}
```

//class testinterest

```
import static org.junit.Assert.assertEquals;
import org.junit.Before;
import org.junit.Test;
public class testinterest {

   @Before
   public void setUp() {
    }
    // Test case for multiplication of all positive
   @Test
   public void testall_Positive() {

      double result=simple_interest.CalculateSimpleInterest(50000.00, 4.0,8.0);
      assertEquals(16000, result,0.001);
   }
// Test case for multiplication of two positive one negetive
```

```
@Test
 public void testtwo_Positive() {

    double result=simple_interest.CalculateSimpleInterest(50000.00, -4.0,8.0);
    assertEquals(16000, result,0.001);
 }
// Test case for multiplication of two negetive one positive
 @Test
 public void testtwo_Negative() {
    double result=simple_interest.CalculateSimpleInterest(-50000.00, -4.0,8.0);
    assertEquals(16000, result,0.001);
 }
 // Test case for multiplication involving zero
 @Test
 public void test_ZeroPositive() {
    double result=simple_interest.CalculateSimpleInterest(0, 4.0,8.0);
    assertEquals(16000, result,0.001);
 }
```

Practical 10

To study and perform validation testing.

Validation testing is a type of software testing that ensures the product meets the user's needs and requirements. It confirms that the software behaves as expected in a real-world scenario and answers the question, *"Are we building the right product?"*

Validation testing is performed during or at the end of the development process to verify the system's functionality from the user's perspective. It typically involves methods like functional testing, system testing, and acceptance testing.

Key Aspects of Validation Testing:

1. **User-Centered**: Focuses on whether the product fulfills the user's requirements.
2. **Real-World Scenarios**: Simulates actual use cases to ensure functionality.
3. **High-Level Testing**: Typically comes after verification testing, which checks whether the product is built correctly.

Example of Validation Testing

Scenario: A team develops a mobile banking app that allows users to check balances, transfer funds, and pay bills.

- **Test Case:** Validate the fund transfer functionality.
- **Steps**:
    1. Open the app and log in with valid credentials.
    2. Navigate to the "Transfer Funds" section.
    3. Select the "From Account" and "To Account."
    4. Enter an amount to transfer and submit.
- **Expected Result:** The app should successfully transfer the entered amount and display a confirmation message.
- **Validation Check:** Ensure the transfer is reflected in the account balances correctly and that the user receives an email confirmation. The transfer process must meet user expectations (like speed and ease of use) and business rules.

Popular Tools for validation testing

1. **Selenium**: For web application validation.
2. **Katalon Studio**: Supports web, API, and mobile validation.
3. **Appium**: Used for validating mobile applications.
4. **Postman (for API Validation):** used for API validation testing
5. **Cypress:** For web application validation.

Practical 11

To study and perform performance based testing.

Performance testing is a type of software testing that evaluates how a system performs under various conditions. The primary focus is to check the speed, responsiveness, scalability, and stability of an application under a specific workload.

Performance testing ensures that the system behaves as expected in terms of:

- **Response time**: How fast the system responds to a user's request.
- **Throughput**: The number of transactions processed in a given time period.
- **Scalability**: The system's ability to handle increased load.
- **Stability**: How the system performs under stress for a prolonged period.

Types of Performance Testing

1. **Load Testing**: Measures system performance under expected user loads.
2. **Stress Testing**: Puts the system under extreme conditions to see how it behaves under stress or when it exceeds its limits.
3. **Spike Testing**: Tests how the system reacts to sudden increases in user load.
4. **Endurance Testing**: Checks how the system behaves when subjected to high loads over extended periods (also called soak testing).
5. **Scalability Testing**: Evaluates the system's ability to scale up or down based on user demand.

Example of Performance Testing

Scenario: A retail website experiences high traffic during flash sales. The development team wants to ensure that the website performs well under such conditions.

- **Test Case**: Load testing for the website's checkout process.
- **Steps**:
    1. Simulate 100, 500, 1,000, and 5,000 concurrent users accessing the checkout page.
    2. Each simulated user adds items to their cart and attempts to make a purchase.
    3. Monitor system metrics such as response time, CPU usage, memory consumption, and transaction success rate.
- **Expected Outcome**:
    1. The website should handle up to 1,000 concurrent users without any noticeable performance degradation.
    2. For 5,000 concurrent users, response time might slightly increase, but the system should still function without crashing.
- **Metrics Monitored**:

- **Response Time**: Should be under 2 seconds for up to 1,000 users.
- **Throughput**: The system should process at least 50 transactions per second.
- **CPU/Memory Usage**: Should not exceed 80% utilization.

Popular Tools for Performance Testing

1. **Apache JMeter**: Open-source tool for load and performance testing of web applications.
2. **LoadRunner**: Commercial tool by Micro Focus, widely used for large-scale performance testing.
3. **Gatling**: Open-source tool designed for web applications, focusing on scalability testing.
4. **Neoload**: Tool for continuous performance testing integrated into DevOps environments.

Practical 12

To study and perform regression based testing.

Regression testing is a type of software testing that ensures recent code changes haven't negatively impacted the existing functionality of the application. The goal is to verify that previously developed and tested features still work as expected after modifications like bug fixes, feature enhancements, or system upgrades.

Key Objectives of Regression Testing:

1. **Detect Unintended Side Effects**: Ensure that changes in one part of the system do not break other parts.
2. **Maintain Stability**: Keep the system stable by validating that old features function as intended after new updates.
3. **Revalidate Existing Functionality**: Confirm that previously working features continue to operate correctly after code changes.

When is Regression Testing Performed?

- After bug fixes.
- After enhancements or new feature additions.
- Following performance or security patches.
- During integration of new modules or systems.

Types of Regression Testing

1. **Corrective Regression**: Testing when no changes are made to the existing system, and existing test cases can be reused.
2. **Retest-all Regression**: Retesting all the tests in the application to ensure no functionality is broken.
3. **Selective Regression**: Testing specific parts of the application that might have been affected by the changes.
4. **Progressive Regression**: Used when new test cases are developed during the modification process.

Example of Regression Testing

Scenario: A team adds a "Wishlist" feature to an e-commerce website that previously allowed only adding items to a cart and purchasing them.

- **Test Case**: Ensure the "Add to Cart" and "Purchase" functionalities still work after introducing the "Wishlist" feature.
- **Steps**:

1. Add an item to the cart and proceed with the checkout process.
2. Ensure the payment gateway redirects properly and the purchase completes without errors.
3. Add the same item to the wishlist and verify that it appears correctly.

- **Expected Outcome**:
  - The existing "Add to Cart" and "Purchase" functionalities should work as before.
  - The new "Wishlist" feature should not interfere with the cart and purchase processes.
- **Regression Check**: The core e-commerce features like adding to the cart, payment, and order confirmation should remain unaffected.

Tools for Regression Testing

1. **Selenium**: An open-source tool used for automating web applications for regression testing.
2. **TestComplete**: A commercial tool for automating desktop, web, and mobile applications.
3. **QTP/UFT**: Unified Functional Testing by Micro Focus for automating functional and regression tests.
4. **Rational Functional Tester (RFT)**: IBM's tool for functional and regression testing.
5. **Katalon Studio**: An easy-to-use tool for automating regression tests across web, mobile, and desktop platforms.