EX. NO: 1

Develop the test plan for testing an e-commerce web/mobile application (www.amazon.in)

Aim:

To create a comprehensive test plan for testing an e-commerce web/mobile application like www.amazon.in requires careful consideration of different testing aspects.

1. Introduction:

- Overview of the application (www.amazon.in)
- Purpose and objectives of the test plan
- Scope of testing (devices, browsers, operating systems, etc.)

2. Test Strategy:

- Testing approach (manual, automation, or a combination)
- Test levels (unit, integration, system, regression, etc.)
- Test types (functional, performance, security, usability, etc.)

3. Test Environment:

- Hardware and software requirements
- Supported browsers and devices
- Test data setup and management

4. Test Cases:

- Detailed test cases covering various scenarios and functionalities
- Test case identification and prioritization
- Test case execution schedule and dependencies

5. Functional Testing:

- Verification of core e-commerce functionalities (e.g., search, product listing, add to cart, checkout, payment, order history, etc.)
- Testing various product categories and attributes
- Testing user registration and account management

6. Usability Testing:

- Evaluation of the user interface (UI) and user experience (UX)
- Ensuring responsiveness across different devices and screen sizes
- Checking accessibility compliance

7. Security Testing:

- Validation of secure data transmission (HTTPS)

- Authentication and authorization testing
- Protection against common security vulnerabilities (e.g., SQL injection, cross-site scripting)

8. Performance Testing:

- Load testing to assess website/mobile app performance under various user loads
- Stress testing to determine system limits and responsiveness during peak traffic
- Response time, resource utilization, and server performance analysis

9. Compatibility Testing:

- Verification of application behavior across different browsers (Chrome, Firefox, Safari, etc.)
- Testing on various operating systems (Windows, macOS, Android, iOS)
- Testing on different devices (desktop, tablets, smartphones)

10. Mobile App Testing:

- App installation and uninstallation testing
- App compatibility with different device configurations
- Testing of app-specific features (push notifications, location services, etc.)

11. Payment Gateway Testing:

- Testing different payment methods (credit card, debit card, net banking, etc.)
- Verification of secure payment transactions

12. Order Fulfillment Testing:

- Testing order processing and shipment tracking
- Validation of order cancellation and refund processes

13. Error Handling and Recovery Testing:

- Validation of error messages and user guidance
- Recovery testing to check the application's behavior after failure scenarios

14. Regression Testing:

- Testing after each update or change to ensure existing functionality remains intact

15. User Acceptance Testing (UAT):

- Involving real users to validate the application's suitability for their needs
- Gathering feedback and making necessary improvements

16. Test Reporting:

- Defect reporting, tracking, and resolution process

- Test execution status and progress reporting
- Test summary report with overall findings and recommendations

17. Risk Assessment:

- Identifying potential risks and their mitigation strategies
- Prioritizing testing efforts based on risk impact

18. Test Deliverables:

- List of documents and artifacts to be produced during testing
- Test data and environment setup documentation

19. Test Exit Criteria:

- Conditions that must be met before concluding testing
- Criteria for approving the application for production release

20. Test Schedule:

- Timeframe and milestones for each testing phase
- Resource allocation and responsibilities

It's important to note that the test plan should be tailored to the specific requirements and characteristics of www.amazon.in or any other e-commerce application under test. Additionally, collaboration between developers, testers, and other stakeholders is crucial throughout the testing process to ensure a successful outcome.

EX. NO: 2

Design the test cases for testing the e-commerce application

Aim:

To design test cases for testing an e-commerce application like www.amazon.in involves creating detailed scenarios that cover various functionalities, user interactions, and edge cases.

1. User Registration:

- Verify successful registration with valid user details (name, email, password).
- Verify error messages for invalid or incomplete user details.
- Check if email address is unique and not already registered.
- Verify that the user receives a verification email after registration.
- Test password strength requirements during registration.

2. User Login:

- Verify successful login with valid credentials.
- Verify error messages for incorrect or invalid login credentials.
- Test "Remember Me" functionality on the login page.
- Check for session timeout and automatic logout after a period of inactivity.

3. Product Search and Listing:

- Test search functionality with valid and invalid keywords.
- Verify the correct display of search results with relevant products.
- Check filtering options (e.g., by category, price, brand) and their accuracy.
- Ensure pagination works correctly and displays the appropriate number of items per page.

4. Product Details Page:

- Verify product details accuracy (name, price, description, images).
- Test "Add to Cart" functionality for different product variations (size, color, quantity).
- Check product availability and out-of-stock scenarios.

5. Shopping Cart:

- Verify products are added to the cart correctly.
- Test updating and removing items from the cart.
- Ensure the cart calculates the total price accurately.
- Test "Proceed to Checkout" functionality.

6. Checkout Process:

- Test different payment methods (credit card, debit card, net banking).
- Verify successful order placement with valid payment details.
- Test the "Cash on Delivery" option (if available).
- Check for validation of invalid payment details.

7. Order History and Tracking:

- Verify the display of the user's order history.
- Test order status updates and shipment tracking.
- Verify the user's ability to cancel an order.

8. User Account Management:

- Test updating user profile information (address, phone number, etc.).
- Verify password change functionality.
- Test account deletion (if applicable) with proper confirmation.

9. Security Testing:

- Test the application for SQL injection vulnerabilities.
- Check for cross-site scripting (XSS) vulnerabilities.
- Verify secure data transmission (HTTPS) during sensitive transactions.
- Test for protection against session fixation attacks.

10. Performance Testing:

- Test the application's response time under different load levels.
- Verify server performance during peak traffic.
- Check the application's ability to handle a large number of concurrent users.

11. Mobile App Specific Test Cases (if applicable):

- Test app installation and uninstallation on various devices.
- Verify app behavior during interruptions (calls, notifications).
- Test screen orientation and device rotation.

12. Usability Testing:

- Verify the user interface for consistency and aesthetics.
- Test the application on different screen sizes and resolutions.
- Check if the application is accessible to users with disabilities.

These are just a few examples of test cases for an e-commerce application. Depending on the application's complexity and specific requirements, you would need to create more test cases to cover all aspects thoroughly. Additionally, test cases may involve both positive and negative scenarios to ensure proper handling of different user interactions and potential errors.

EX. NO: 3

Test the e-commerce application and report the defects in it

Aim:

To test the e-commerce application (www.amazon.in) and report defects, follow these steps:

1. Preparation:

- Set up the test environment with the required hardware, software, and test data.
- Identify the browsers, devices, and operating systems you will be testing on.
- Familiarize yourself with the application's features and functionalities.

2. Execute Test Cases:

- Execute the test cases designed in the previous step for each functionality.
- Record the test results, including any defects or issues encountered during testing.
- Include screenshots or videos if necessary to provide visual evidence of defects.

3. Functional Testing:

- Verify the core functionalities, such as product search, product listing, cart, checkout, and payment.
- Ensure that user registration, login, and account management work as expected.

4. Usability Testing:

- Evaluate the user interface and user experience for ease of use and navigation.
- Test the responsiveness of the application across different devices.

5. Security Testing:

- Verify secure data transmission (HTTPS) during transactions and sensitive operations.
- Test for potential security vulnerabilities like SQL injection and cross-site scripting (XSS).

6. Performance Testing:

- Test the application's response time under different load levels.
- Check for performance issues during peak traffic.

7. Compatibility Testing:

- Test the application on various browsers, devices, and operating systems to ensure cross-platform compatibility.

8. Payment Gateway Testing:

- Test different payment methods (credit card, debit card, net banking) for successful transactions.
- Verify error handling for invalid payment details.

9. Order Fulfillment Testing:

- Test order processing and shipment tracking.
- Verify order cancellation and refund processes.

10. **User Acceptance Testing (UAT):**
    - Involve real users to test the application and gather feedback.
11. **Defect Reporting:**
    - Document all defects found during testing.
    - Provide detailed steps to reproduce each defect.
    - Include information about the browser, device, and operating system used during testing.
    - Assign a severity level (critical, high, medium, low) and priority to each defect.
12. **Defect Tracking:**
    - Use a defect tracking tool to log and manage defects.
    - Keep track of the status of each defect (open, assigned, fixed, closed, etc.).
13. **Reporting and Communication:**
    - Prepare a comprehensive test report summarizing the testing activities and results.
    - Include an overview of the application's overall quality, highlighting critical issues.
    - Share the test report with the development team and stakeholders.
14. **Retesting and Regression Testing:**
    - After the defects are fixed, retest them to ensure they are resolved.
    - Perform regression testing to ensure that new changes do not introduce new defects.

Remember, effective communication and collaboration between the testing team, development team, and stakeholders are crucial for a successful testing process. Defects should be reported in a clear and concise manner to facilitate efficient resolution.

EX. NO: 4

Develop the test plan and design the test cases for an Inventory control system

Test Plan for Inventory Control System:

1. Introduction:

- Overview of the Inventory Control System.
- Purpose and objectives of the test plan.
- Scope of testing (modules, functionalities, etc.).
- Test environment details (hardware, software, databases).

2. Test Strategy:

- Testing approach (manual, automation, or a combination).
- Test levels (unit, integration, system, acceptance).
- Test types (functional, non-functional, security, usability).

3. Test Scope and Deliverables:

- In-scope and out-of-scope items for testing.
- List of test deliverables (test cases, test data, test reports).

4. Test Environment:

- Hardware and software requirements.
- Database setup and configuration.
- Test data preparation and management.

5. Test Cases Design:

- Test case identification and numbering.
- Test case description and steps.
- Expected results and acceptance criteria.
- Test data requirements for each test case.

6. Functional Testing:

- Inventory data entry and validation testing.
- Testing inventory updates (addition, deletion, modification).
- Verification of reorder level and reorder quantity calculation.
- Testing stock transfers between locations or warehouses.

7. Integration Testing:

- Verify data synchronization between the inventory system and other integrated systems (e.g., accounting, purchase, sales).
- Test data flow and integrity during data exchange.

8. System Testing:

- End-to-end testing of the entire inventory system.
- Test scenarios involving multiple modules and user interactions.
- Testing various user roles and permissions.

9. Non-Functional Testing:

- Performance testing to ensure system responsiveness under different load levels.
- Security testing to validate access controls and data protection.
- Usability testing to assess the user-friendliness of the system.

10. User Acceptance Testing (UAT):

- Involving end-users to validate the system's suitability for their needs.
- Gathering feedback and making necessary improvements.

11. Error Handling and Recovery Testing:

- Test error messages and proper handling of exceptions.
- Verify the system's ability to recover from unexpected failures.

12. Reports and Documentation Testing:

- Testing various inventory reports (stock status, reorder reports, etc.).
- Verification of user manuals and documentation.

13. Risk Assessment:

- Identify potential risks and their mitigation strategies.
- Prioritize testing efforts based on risk impact.

14. Test Schedule and Resource Allocation:

- Timeframe and milestones for each testing phase.
- Allocation of resources (testers, equipment) for testing.

15. Test Execution and Defect Reporting:

- Execute test cases as per the test schedule.
- Report defects using a defect tracking tool.
- Include necessary details for each defect (steps to reproduce, severity, priority).

16. Test Closure:

- Evaluation of test completion criteria.
- Lessons learned and suggestions for improvement.

Sample Test Cases for Inventory Control System:

1. Inventory Data Entry:

- Test if the system allows adding new inventory items with valid details.
- Verify that mandatory fields are properly validated during data entry.
- Test if duplicate inventory items are not allowed.

2. Inventory Update:

- Test inventory quantity update when goods are received through purchase orders.
- Verify that the stock level decreases after goods are sold.
- Test the system's response when inventory quantity goes below the reorder level.

3. Reorder and Reorder Quantity Calculation:

- Verify that the reorder level and reorder quantity are correctly set for each inventory item.
- Test the system's response when the stock level reaches the reorder point.

4. Stock Transfer:

- Test the ability to transfer stock between different locations or warehouses.
- Verify that the stock quantity is correctly adjusted in both source and destination locations.

5. Integration with Accounting System:

- Test if inventory data is correctly synchronized with the accounting system.
- Verify that inventory-related financial transactions are accurately recorded.

6. Performance Testing:

- Test system response time during peak load (e.g., multiple users accessing the system simultaneously).
- Verify that the system can handle a large number of inventory items without performance degradation.

7. Security Testing:

- Verify that access to inventory data is restricted based on user roles and permissions.
- Test for SQL injection and other security vulnerabilities.

8. Usability Testing:

- Evaluate the user interface for ease of use and intuitive navigation.
- Test if the system provides appropriate feedback and error messages.

9. Reports Testing:

- Test inventory reports for accuracy and completeness.
- Verify that report data matches the actual inventory status.

10. Error Handling and Recovery Testing:

- Test the system's response to invalid data entry and input errors.
- Verify that the system recovers gracefully after unexpected failures.

These are just a few examples of test cases for an Inventory Control System. Depending on the complexity and requirements of the system, you may need to create additional test cases to cover all aspects of the application. Each test case should have clear steps, expected results, and acceptance criteria to ensure thorough testing and accurate defect reporting.

EX. NO: 5

Execute the test cases against a client server or Desktop application and identify the defects.

1. Set Up Test Environment:

- Install the client application on the test machines.
- Set up the server environment with the necessary databases and services.
- Ensure that the test environment closely resembles the production environment.

2. Test Data Preparation:

- Prepare test data that covers various scenarios, including boundary cases and negative test scenarios.
- Use a mix of valid and invalid data to ensure thorough testing.

3. Execute Test Cases:

- Start executing the test cases you designed earlier.
- Record the test results, including pass/fail status and any observations or comments.

4. Functional Testing:

- Verify that the application's core functionalities work as expected.
- Test all the features and functions according to the test cases.

5. Integration Testing:

- Test the interactions between the client and server components.
- Verify data synchronization and data integrity.

6. Performance Testing:

- Measure the application's performance, such as response time and resource usage.
- Check the client-server communication efficiency under different load levels.

7. Security Testing:

- Test for any security vulnerabilities, like unauthorized access or data breaches.
- Verify data encryption and secure communication between client and server.

8. Usability Testing:

- Evaluate the user interface and user experience.
- Check if the application is intuitive and easy to use.

9. Error Handling and Recovery Testing:

- Test the application's response to different types of errors.
- Verify that the application recovers gracefully after unexpected failures.

10. Defect Identification and Reporting:

- Document any defects or issues encountered during testing.
- Include detailed steps to reproduce each defect.

- Assign a severity level (critical, high, medium, low) and priority to each defect.

11. Retesting and Regression Testing:

- After defects are fixed, retest them to ensure they are resolved.
- Perform regression testing to ensure that new changes do not introduce new defects.

12. Test Closure:

- Evaluate the completion criteria for testing.
- Prepare a test summary report with the overall testing results.

Remember to communicate effectively with the development team and stakeholders when reporting defects. Provide all relevant information to help them understand and resolve the issues efficiently.

Executing test cases against a client-server or desktop application requires a systematic and organized approach. It's essential to follow the test plan and document all observations and results accurately. By doing so, you can ensure a successful testing process and a more reliable application.

EX. NO: 6

Test the performance of the e-commerce application

Aim:

Testing the performance of an e-commerce application is crucial to ensure that it can handle the expected user load and provide a smooth user experience. Performance testing helps identify bottlenecks and weaknesses in the application, enabling improvements before deployment. Steps to test the performance of the e-commerce application:

1. Identify Performance Metrics:

- Determine the key performance metrics to measure, such as response time, throughput, concurrency, and resource utilization.

2. Performance Test Environment Setup:

- Create a separate performance test environment that resembles the production environment but is isolated for testing.
- Ensure that the test environment has adequate hardware resources, including servers and databases.

3. Define Performance Test Scenarios:

- Identify critical user journeys and scenarios in the e-commerce application, such as product search, add to cart, checkout, and payment processing.
- Create performance test scripts or scenarios based on these user journeys.

4. Load Testing:

- Perform load testing to assess the application's performance under typical and peak user loads.
- Gradually increase the user load to simulate realistic traffic and observe how the system handles the increased load.

5. Stress Testing:

- Conduct stress testing to determine the application's breaking point and understand how it behaves under extreme load conditions.
- Increase the load beyond the application's capacity to identify performance bottlenecks and potential failures.

6. Spike Testing:

- Perform spike testing to evaluate the application's response to sudden spikes in user traffic.
- Create sudden and significant load variations to see if the application can handle unexpected surges.

7. Endurance Testing:

- Conduct endurance testing to assess the application's stability and performance over an extended period.
- Run the test for several hours or even days to identify memory leaks, database connection issues, or resource depletion.

8. Performance Monitoring:

- Monitor the application's performance during the test to collect performance metrics and identify performance hotspots.
- Use performance monitoring tools to gather data on CPU usage, memory consumption, network traffic, and response times.

9. Analyze Results and Identify Bottlenecks:

- Analyze the performance test results to identify performance bottlenecks and areas of improvement.
- Pinpoint slow-performing components, database queries, or network delays.

10. Fixing and Re-testing:

- Work with developers to fix identified performance issues.
- Re-run the performance tests to verify that the fixes have improved the application's performance.

11. Performance Test Reporting:

- Prepare a comprehensive performance test report with the test objectives, methodology, results, and findings.
- Include recommendations for improvement and any outstanding performance concerns.

12. Continuous Performance Testing:

- Implement continuous performance testing as part of the CI/CD pipeline to catch performance regressions in subsequent releases.

13. Real User Monitoring (RUM):

- Implement Real User Monitoring to collect performance data from actual users, allowing you to analyze and optimize the application in real-world scenarios.

By thoroughly testing the performance of the e-commerce application, you can ensure its stability, responsiveness, and scalability, delivering a positive user experience even during peak traffic periods.

EX. NO: 7

Automate the testing of e-commerce applications using Selenium

Aim:

To automate the testing of e-commerce applications using Selenium can significantly improve testing efficiency and provide faster feedback on application quality. Selenium is a widely-used open-source automation testing framework that allows you to automate web browsers for testing web applications. Here's a step-by-step guide to automating e-commerce testing using Selenium:

1. Set Up the Environment:

- Install Java Development Kit (JDK) on your machine.
- Install an Integrated Development Environment (IDE) like Eclipse or IntelliJ.
- Download and configure the latest Selenium WebDriver Java bindings.

2. Identify Test Scenarios:

- Identify the critical test scenarios in the e-commerce application that you want to automate.
- Select test cases covering key functionalities like login, search, add to cart, checkout, etc.

3. Set Up Test Framework:

- Choose a testing framework to work with Selenium, such as TestNG or JUnit.
- Set up the testing framework in your IDE and configure it with Selenium WebDriver.

4. Write Test Scripts:

- Start writing test scripts using Java to automate the identified test scenarios.
- Use Selenium WebDriver commands to interact with web elements like clicks, inputs, and assertions.

5. Enhance Test Scripts with Page Objects:

- Implement the Page Object Model (POM) design pattern to improve test script maintainability and reusability.
- Create separate Java classes representing each page of the e-commerce application with corresponding web elements and methods.

6. Add Assertions and Verifications:

- Include assertions in the test scripts to validate that the expected outcomes match the actual results.
- Verify that the correct products are displayed, the cart is updated, and the payment process is successful.

7. Handle Synchronization and Waits:

- Use explicit and implicit waits to handle synchronization issues and ensure that the web elements are loaded before interacting with them.

8. Execute Test Suites:

- Organize the test scripts into test suites based on functionalities or modules.
- Run the test suites using the testing framework in your IDE.

9. Generate Test Reports:

- Integrate a reporting library like Extent Reports to generate detailed test reports with pass/fail status and screenshots.

10. Set Up Continuous Integration (CI):

- Integrate the automated tests into a CI/CD pipeline using tools like Jenkins or Azure DevOps.
- Schedule automated test runs for every build or deployment.

11. Handle Test Data:

- Use external test data files or databases to handle test data for different test scenarios.

12. Handle Browser Compatibility:

- Execute tests on different browsers (Chrome, Firefox, Safari, etc.) and versions to ensure cross-browser compatibility.

13. Handle Mobile Testing (Optional):

- Use Appium, an extension of Selenium, to automate testing on mobile devices for mobile versions of the e-commerce application.

14. Perform Continuous Maintenance:

- Keep the automated test scripts up-to-date with application changes.
- Regularly review and update the test scripts as the e-commerce application evolves.

Automating e-commerce testing using Selenium can streamline the testing process, improve test coverage, and accelerate feedback, allowing you to deliver a more reliable and robust e-commerce application.

EX. NO: 8

Integrate TestNG with the above test automation.

Aim:

To integrate TestNG with the Selenium test automation framework described above, follow these steps:

1. Set Up TestNG:

- Ensure you have installed TestNG in your Eclipse or IntelliJ IDE. If not, you can install TestNG using the following steps:
  - In Eclipse, go to "Help" -> "Eclipse Marketplace."
  - Search for "TestNG" in the Marketplace and click "Go."
  - Click "Install" next to the TestNG item to install the plugin.

2. Organize TestNG Test Suites:

- Organize your test scripts into TestNG test classes and test methods. Each test class represents a page or module, and each test method represents a specific test scenario.

3. Add TestNG Annotations:

- Add TestNG annotations to your test classes and test methods to control the test execution flow. Common annotations include:
  - **@Test**: To mark test methods that need to be executed.
  - **@BeforeTest**: To set up preconditions before executing the test methods.
  - **@AfterTest**: To perform cleanup or tear down tasks after executing the test methods.
  - **@DataProvider**: To provide test data to parameterized test methods.

4. Run TestNG Test Suites:

- Right-click on the test class or test XML file that contains the TestNG test suites.
- Select "Run As" -> "TestNG Test" to execute the test suites.

5. View TestNG Reports:

- TestNG generates detailed HTML test reports, including test results, pass/fail status, and execution times.
- By default, the reports are saved in the **test-output** folder in the project directory.

Sample TestNG Integration:

Below is a sample TestNG integration with Selenium for an e-commerce application:

```
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
public class EcommerceTest {

   // Set up WebDriver and other preconditions
   @BeforeTest
```

```
    public void setup() {
        // Your setup code here
    }

    // Test method 1
    @Test
    public void testLogin() {
        // Your test script for login functionality
    }

    // Test method 2
    @Test
    public void testSearchProduct() {
        // Your test script for searching a product
    }

    // Test method 3
    @Test
    public void testAddToCart() {
        // Your test script for adding a product to the cart
    }

    // Test method 4
    @Test
    public void testCheckout() {
        // Your test script for the checkout process
    }

    // Clean up WebDriver and other post-test tasks
    @AfterTest
    public void cleanup() {
        // Your cleanup code here
    }
}
```

With TestNG annotations added to the test classes, you can easily execute the test scripts and obtain detailed reports of test results. Additionally, TestNG allows you to perform parameterized testing using data providers, which can be beneficial for testing various scenarios with different test data sets. Integration of TestNG with Selenium provides a powerful and flexible framework for automating and managing your e-commerce application tests efficiently.