Regression Testing - Software Testing

Regression Testing involves re-executing a previously created test suite to verify that recent code changes haven't caused new issues. This verifies that updates, bug fixes, or enhancements do not break the functionality of the application.

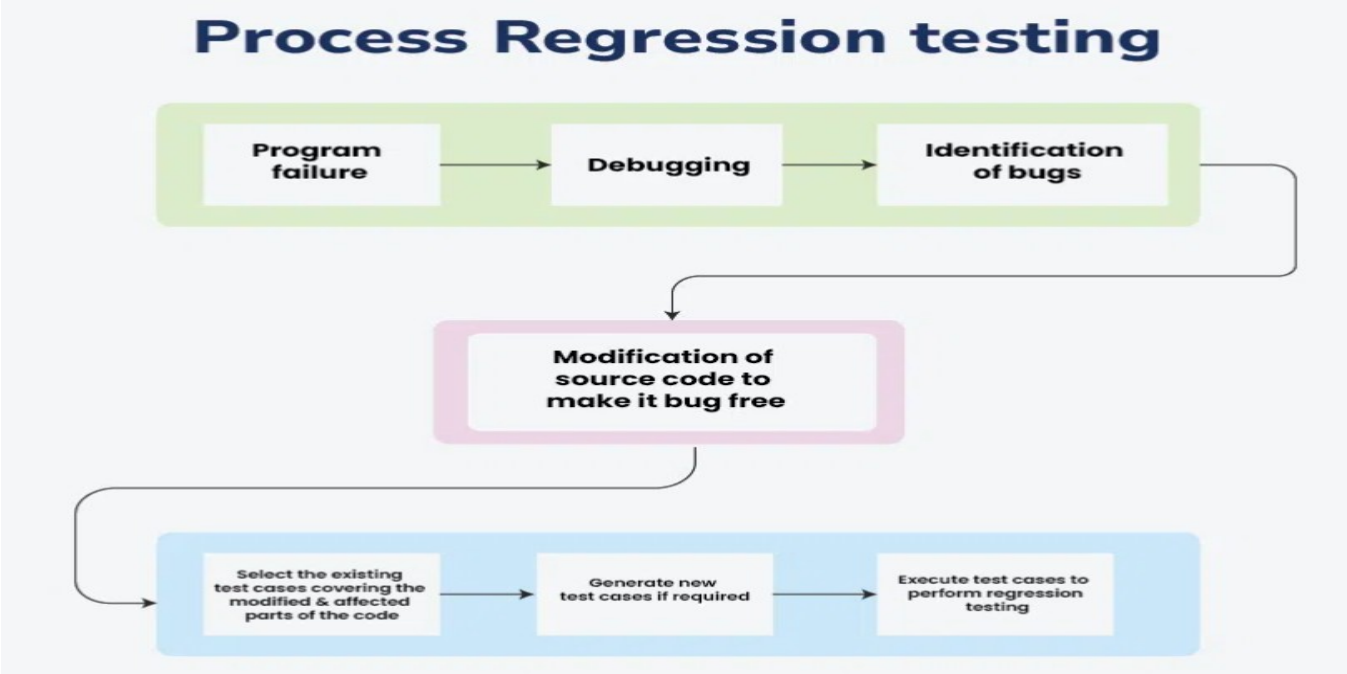
When to do Regression Testing?

Regression testing is necessary in several scenarios to maintain software quality:

- When new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.

Process of Regression Testing

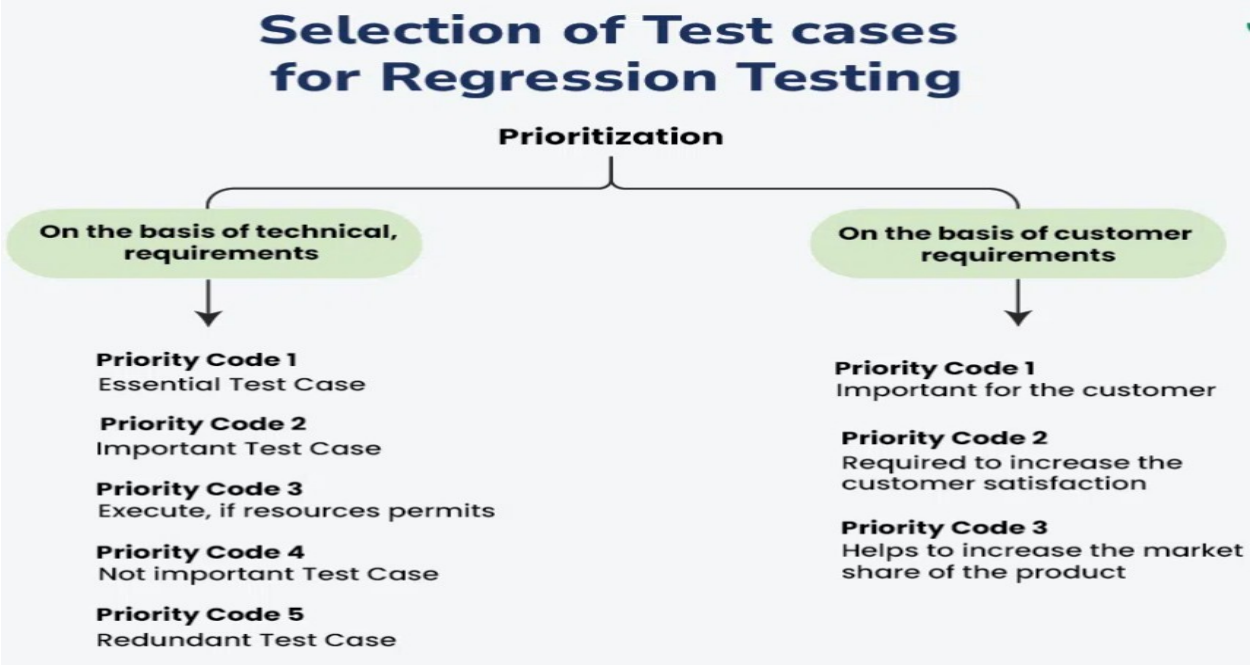
Here is the step-by-step process of the regression testing:



1. **Identify Code Changes**: Analyze the source code to determine which areas have been modified, such as new features, bug fixes, or optimizations.
2. **Debug and Fix Failures**: If existing test cases fail due to changes, debug the code to identify and resolve defects.
3. **Modify Code**: Apply necessary updates to the code to incorporate changes or fixes.
4. **Select Test Cases**: Choose relevant test cases from the existing test suite that cover modified and affected areas. Add new test cases if needed to address new functionality.
5. **Execute Regression Tests**: Run the selected test cases, either manually or using automated tools, to verify system behavior.
6. **Analyze Results**: Review test outcomes to identify regressions, document issues, and recommend fixes.
7. **Retest as Needed**: If defects are found, fix them and re-run tests to confirm resolution.

Techniques for Selecting Test Cases for Regression Testing

Selecting the right test cases is critical for efficient regression testing. Common techniques include:



- **Select all test cases:** In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not very efficient.
- **Select test cases randomly:** In this technique, test cases are selected randomly from the existing test suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- **Select modification traversing test cases:** In this technique, only those test cases are selected that cover and test the modified portions of the source code and the parts that are affected by these modifications.
- **Select higher priority test cases:** In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The test case with the highest priority has the highest rank. For example, a test case with priority code 2 is less important than a test case with priority code.

Regression Testing Example

E-Commerce Website Core Functionality: In this regression test, we will check:

1. Login functionality.

2. Add to Cart functionality.
3. Logout functionality.

Regression testing ensures that no new changes or fixes break the existing system.

Basetest.java

```
package Test;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;

public class BaseTestMain {

    protected WebDriver driver;
    protected String Url = "https://ecommerce.artoftesting.com/";

    // Set up the ChromeDriver
    @BeforeMethod
    public void setup() {
        // Set the path to your chromedriver executable
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\path of the chromedriver\\drivers\\chromedriver.exe");


        // Initialize the ChromeDriver
        driver = new ChromeDriver();
    }

    // Close the browser after each test
    @AfterMethod
    public void teardown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

1. Test Login Functionality:

Verify that the user can still log in successfully after updates.

Test Steps:

1. Open the login page.
2. Input valid username and password.
3. Submit the login form.
4. Verify the user is redirected to the correct URL (i.e., logged in successfully).
5. Verify that the user session has been created (this can be done optionally by checking cookies, or session IDs).

LoginPageTest.java

```
package ArtOfTesting;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import Test.BaseTestMain;

public class LoginPageTest extends BaseTestMain {

  @Test
  public void TestLogin() {
    // Step 1: Navigate to the login page
    driver.get(Url);

    // Step 2: Enter valid username and password
    driver.findElement(By.name("uname")).clear();
    driver.findElement(By.name("uname")).sendKeys("auth_user");

    driver.findElement(By.name("pass")).clear();
    driver.findElement(By.name("pass")).sendKeys("auth_password");

    // Step 3: Click on the Login button
    driver.findElement(By.className("Login_btn__pALc8")).click();

    // Step 4: Verify successful login by checking the URL
```

```
        Assert.assertEquals(driver.getCurrentUrl(),
"https://ecommerce.artoftesting.com/");

        System.out.println("Login Successful");
    }
}
```

2. Test Add to Cart Functionality:

Ensure that users can successfully add products to the cart and the cart updates accordingly.

Test Steps:

1. Log in (reuse the login test).
2. Select a product to add to the cart.
3. Click the "Add to Cart" button.
4. Verify that the cart is updated with the added item.
5. Optionally, check if the correct number of items is displayed in the cart.

AddToCartTest.java

```
package ArtOfTesting;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.testng.Assert;
import org.testng.annotations.Test;
import Test.BaseTestMain;

public class AddToCartTest extends BaseTestMain {

    @Test
    public void TestAddToCart() {
        // Step 1: Log in (reuse the login test)
        driver.get(Url);
        driver.findElement(By.className("Login_btn__pALc8")).click(); // Log in

        // Step 2: Navigate to the product

driver.findElement(By.xpath("/html/body/div/div/div[3]/div/div/select")).c
lick();
```

```
        WebElement filterOption =
driver.findElement(By.className("Header_select__8rhX+"));
        Actions action = new Actions(driver);
        action.sendKeys(filterOption, "Down").perform();
        action.sendKeys("ENTER").perform();

        // Step 3: Select a product and add to cart
        WebElement bookAddition = driver.findElement(By.cssSelector("#root >
div > div.Products_body__ifIXG > div > div:nth-child(1) >
div.Products_quantity__54gJ2 > svg:nth-child(3) > path"));
        action.doubleClick(bookAddition).perform();
        driver.findElement(By.cssSelector("#root > div >
div.Products_body__ifIXG > div > div:nth-child(1) >
div.Products_priceSection__j7qrQ > button")).click();

        // Step 4: Check if the cart is updated (URL check)
        driver.findElement(By.className("Header_cart__Jnfkn")).click();
        String cartURL = "https://ecommerce.artoftesting.com/cart";
        String currentURL = driver.getCurrentUrl();
        Assert.assertEquals(currentURL, cartURL);

        System.out.println("Item successfully added to the cart.");
    }
}
```

3. Test Logout Functionality:

Ensure that users can log out successfully and are redirected to the login page.

Test Steps:

1. Log in (reuse the login test).
2. Click the logout button.
3. Verify that the user is logged out and redirected to the login page.

LogoutTest.java

```
package ArtOfTesting;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.testng.Assert;
```

```
import org.testng.annotations.Test;
import Test.BaseTestMain;

public class LogoutTest extends BaseTestMain {

   @Test
   public void TestLogout() {
      // Step 1: Log in (reuse the login test)
      driver.get(Url);
      driver.findElement(By.name("uname")).clear();
      driver.findElement(By.name("uname")).sendKeys("auth_user");

      driver.findElement(By.name("pass")).clear();
      driver.findElement(By.name("pass")).sendKeys("auth_password");

      driver.findElement(By.className("Login_btn__pALc8")).click();

      // Step 2: Log out by clicking the logout button
      WebElement logoutButton =
driver.findElement(By.xpath("/html/body/div/div/div[1]/div/div[2]/button
/div/span"));
      Actions actions = new Actions(driver);
      actions.doubleClick(logoutButton).perform();

      // Step 3: Verify user is logged out and redirected to login page
      Assert.assertEquals(driver.getCurrentUrl(),
"https://ecommerce.artoftesting.com/login");

      System.out.println("Logout Successful");
   }
}
```

Running the Regression Tests:

Once the individual tests for login, add to cart, and logout are written, you can combine them into a regression test suite.

RegressionTestSuite.java

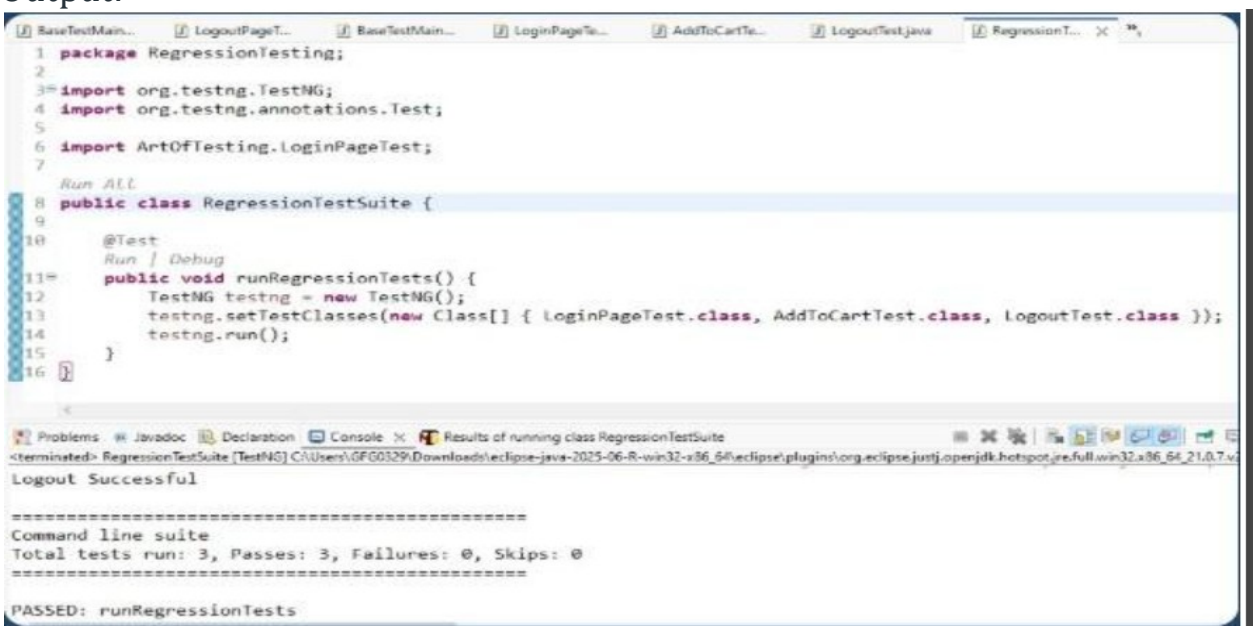
```
package TestSuite;

import ArtOfTesting.LoginPageTest;
```

```
import ArtOfTesting.AddToCartTest;
import ArtOfTesting.LogoutTest;
import org.testng.annotations.Test;
import org.testng.TestNG;

public class RegressionTestSuite {

   @Test
   public void runRegressionTests() {
      TestNG testng = new TestNG();
      testng.setTestClasses(new Class[] { LoginPageTest.class,
AddToCartTest.class, LogoutTest.class });
      testng.run();
   }
}
```

Regression testing aims to ensure that the core features continue to work after any new changes or updates in the system. Here's how this applies to your code:

1. **Login Functionality**: Ensures that the login feature works as expected after any backend or UI changes.
2. **Add to Cart**: Verifies that the user can still add items to the cart and that the cart behaves correctly.
3. **Logout**: Confirms that users can still log out successfully and are redirected properly.

Output:



Regression Testing Tools

In regression testing, we generally select the Test Cases from the existing test suite itself and hence, we need not compute their expected output, and it can be easily automated due to this reason. Automating the process of regression testing will be very effective and time-saving.

The most commonly used tools for regression testing are:

- **Selenium:** Open-source, supports multiple browsers and programming languages, and is widely used for automating web application tests.
- **Ranorex Studio:** A comprehensive solution for testing web, desktop, and mobile applications with both codeless and coded automation options.
- **testRigor:** AI-powered test automation tool that simplifies test creation with natural language processing and requires no coding skills.
- **Sahi Pro:** A user-friendly tool that supports cross-browser testing and integrates well with continuous integration systems like Jenkins.
- **Testlio:** A cloud-based solution with a global network of testers, ideal for on-demand testing, especially for regression testing in real-world scenarios.