

Homework Assignment 6 Part I - CSE 464

Bhavya Patel - ASU ID: 1225740997

Q1: a)

Gray-Box Testing: Gray-box testing combines knowledge of the web application's internal design with tests executed through its external interfaces, so it sits between pure black-box and pure white-box testing in terms of how much implementation detail the tester uses.

Difference from Black Box: While Black Box testing focuses solely on the inputs and outputs without knowing the internal workings, Gray Box testing utilizes knowledge of the system's design and operating environment to design the tests.

Difference from White Box: While White Box testing, which requires full access to the source code to test internal logic, Gray Box testing does not rely entirely on the code but focuses on the application design and the interoperability of system components.

b) Different Aspects of Web Application Testing

Content Testing: When performing Content testing, my primary goal is to identify three specific types of errors: syntactic errors (such as typos), semantic errors, and errors in the content's organization or structure. I also ensure that I check for copyright issues and verify that any internal links are accurate. I can use Selenium here, as it allows me to execute tests that verify dynamically derived content driven by my databases.

Interface Testing: When performing interface testing, I verify that the interface features are presented to the user without errors, ensuring the design rules and aesthetics are met. I treat individual interface mechanisms similarly to unit testing, and I validate the complete interface against specific use cases to catch any semantic errors. I would use Selenium WebDriver to locate elements and perform operations on them.

Navigation Testing: For Navigation testing, my focus here is on ensuring that navigation pathways enable users to navigate through the application logic correctly and that the interaction feels consistent. I need to verify that the app guides user actions effectively and provides meaningful feedback. I would use Selenium WebDriver to handle browser navigation commands, such as moving back and forward or following links.

Component Testing: For Component testing, my focus here is on ensuring that navigation pathways enable users to navigate through the application logic correctly and that the interaction feels consistent. I need to verify that the app guides user actions effectively and provides meaningful feedback. I would use frameworks like SimpleTest for the server-side code, and DBUnit specifically to test database-related functionalities.

Configuration Testing: When performing configuration testing, I need to address the variability of the web environment by testing both server-side and client-side configurations. On the client side, I test compatibility with various hardware, operating systems, browsers, and plug-ins to ensure a consistent user experience, regardless of the user's setup. Selenium Grid is my go-to tool here because it allows me to run my tests in parallel across different environments and browsers.

Performance Testing: When I perform Performance testing, I would check the operating characteristics of the web app to ensure they meet the needs of the end users. This involves testing the server-side environment and network pathways, often by simulating load balancing and high network traffic. To handle this, I would use LoadRunner to simulate load and test the system's performance.

Security Testing: When performing Security testing, my main goal is to prevent unauthorized access to the web app's content and functionality. I specifically test for vulnerabilities, such as SQL Injection, to ensure the database remains secure. While I might use specialized scanners, I can also utilize Selenium to automate input scenarios that test for security flaws, such as SQL injection.

Q2: a) Copy and paste your bmiTest.java program here.

```
import java.time.Duration;
import org.junit.*;
import static org.junit.Assert.*;
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;

public class bmiTest {
    private WebDriver driver;

    @Before
    public void startDriver() {
        System.setProperty("webdriver.chrome.driver",
        "/Users/bhavyapatel/Documents/Fall25/CSE464/chromedriver-mac-arm64/chromedriver");
        driver = new ChromeDriver();

        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
    }

    // Case 1: Underweight interior point (BMI well below 18.5)
    @Test
    public void testcase1() {
        verifyBMI("70", "100", "14.35",
        "Underweight");
    }

    // Case 2: Normal weight interior point (BMI clearly between 18.5 and 25)
```

```
@Test
public void testcase2() {
    verifyBMI("70", "150", "21.52", "Normal
weight");
}

// Case 3: Overweight interior point (BMI clearly
between 25 and 30)
@Test
public void testcase3() {
    verifyBMI("70", "180", "25.82", "Overweight");
}

// Case 4: Obese interior point (BMI clearly
greater than or equal to 30)
@Test
public void testcase4() {
    verifyBMI("70", "220", "31.56", "Obese");
}

// Case 5: Boundary just below 18.5 (should still
be Underweight)
@Test
public void testcase5() {
    verifyBMI("70", "128", "18.37",
"Underweight");
}

// Case 6: Boundary at/just above 18.5 (first
Normal weight value)
@Test
public void testcase6() {
```

```
        verifyBMI("70", "129", "18.51", "Normal
weight");
    }

    // Case 7: Boundary just below 25 (last Normal
weight value)
    @Test
    public void testcase7() {
        verifyBMI("70", "174", "24.96", "Normal
weight");
    }

    // Case 8: Boundary at/just above 25 (first
Overweight value)
    @Test
    public void testcase8() {
        verifyBMI("70", "175", "25.11", "Overweight");
    }

    // Case 9: Boundary just below 30 (last Overweight
value)
    @Test
    public void testcase9() {
        verifyBMI("70", "209", "29.99", "Overweight");
    }

    // Case 10: Boundary at/just above 30 (first Obese
value)
    @Test
    public void testcase10() {
        verifyBMI("70", "210", "30.13", "Obese");
    }
```

```
// Case 11: Negative height should trigger the
'negative' error message

@Test
public void testcase11() {
    verifyBMI("-10", "150", null, "Height or
weight cannot be negative");
}

// Case 12: Negative weight should trigger the
'negative' error message

@Test
public void testcase12() {
    verifyBMI("70", "-150", null, "Height or
weight cannot be negative");
}

// Case 13: Zero height should trigger the 'zero'
error message

@Test
public void testcase13() {
    verifyBMI("0", "150", null, "Height or weight
cannot be zero");
}

// Case 14: Zero weight should trigger the 'zero'
error message

@Test
public void testcase14() {
    verifyBMI("70", "0", null, "Height or weight
cannot be zero");
}
```

```
    private void verifyBMI(String height, String
weight, String expectedBMI, String expectedMessage) {

driver.get("http://webstrar100.fulton.asu.edu/page2/")
;

        WebElement hInput =
driver.findElement(By.id("H"));
        WebElement wInput =
driver.findElement(By.id("W"));
        WebElement calcBtn =
driver.findElement(By.id("Button1"));

        hInput.clear();
        hInput.sendKeys(height);
        wInput.clear();
        wInput.sendKeys(weight);

        calcBtn.click();

        WebElement bmiOutput =
driver.findElement(By.id("BMI"));
        WebElement msgOutput =
driver.findElement(By.id("Message"));

        String actualMessage =
msgOutput.getAttribute("value");

assertTrue(actualMessage.contains(expectedMessage));

        if (expectedBMI != null) {
```

```

        String actualBMIStr =  

bmiOutput.getAttribute("value");  

        double expectedVal =  

Double.parseDouble(expectedBMI);  

        double actualVal =  

Double.parseDouble(actualBMIStr);  

        assertEquals(expectedVal, actualVal,  

0.01);  

    }  

}  
  

@After  

public void closeDriver() {  

    driver.quit();  

}  

}

```

b) Upload the bmiTest.java to your Google Drive and share the link (Both part a and part b are needed to earn points for this question)

<https://drive.google.com/file/d/1SNkm2UbyJKKOhAon1QO9Uu2neS7xJA75/view?usp=sharing>

Additionally Screenshot of all 14 test cases passing on my computer:

The screenshot shows an IDE interface with the following details:

- Project:** HW6-P1
- File:** bmiTest.java
- Code Snippet:**

```
public class bmiTest {  
    private WebDriver driver; 9 usages  
  
    @Before  
    public void startDriver() {  
        System.setProperty("webdriver.chrome.driver",  
            "/Users/heavyapatel/Documents/Fall25/CSE464/chromedriver-mac-arm64/chromedriver");  
        driver = new ChromeDriver();  
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));  
    }  
}
```

- Run:** bmiTest
- Test Results:** 14 tests passed, 57 sec 434 ms

testcase	Time
testcase10	4 sec 636 ms
testcase11	4 sec 21 ms
testcase12	4 sec 129 ms
testcase13	4 sec 33 ms
testcase14	4 sec 95 ms
testcase1	4 sec 6 ms
testcase2	4 sec 66 ms
testcase3	4 sec 24 ms
testcase4	4 sec 114 ms
testcase5	3 sec 988 ms
testcase6	4 sec 70 ms
testcase7	4 sec 92 ms
testcase8	4 sec 166 ms
testcase9	3 sec 994 ms

- Logs:** Multiple WARNING messages from org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch indicating Selenium version updates.
- Bottom Status:** HW6-P1 > src > main > java > bmiTest
- Bottom Right:** 131:28 LF UTF-8 4 spaces