Department of
Computer Science and Engineering

Title: Software Testing (Part I: Automation Testing) for a specific project

Integrated Design Project II

CSE 406

Green University of Bangladesh

1 Objective(s)

- To explain the purpose of software testing.
- To describe different types of software testing tools and techniques.
- To construct suitable test cases for automated software testing.

2 Problem Analysis

2.1 What is a Software Testing?

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

2.2 Benefits of Software Testing

Here are the benefits of using software testing-

- **Cost Effective Development:** Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.
- **Product Improvement:** During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.
- **Test Automation:** Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.
- **Quality Check:** Software testing helps in determining following set of properties of any software such as Functionality, Reliability, Usability, Efficiency, Maintainability, Portability etc.

2.3 Types of Software Testing

There are various types of testing available in the market, which are used to test the application or the software. These types are shown in Fig. 1.

- **Manual Testing:** The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing.
- **Automation Testing:** Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

3 Methodology

3.1 [Part-I] Software Testing Tool: JUnit for Automation Testing

Testing is the process of checking the functionality of an application to ensure it runs as per requirements. Unit testing comes into picture at the developers level; it is the testing of single entity (class or method). Unit testing plays a critical role in helping a software company deliver quality products to its customers. Unit testing can be done in two ways Manual Testing and Automated Testing. Software testing tools are required for the betterment of the application or software.

JUnit is a unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively known as xUnit. JUnit promotes the idea of "first

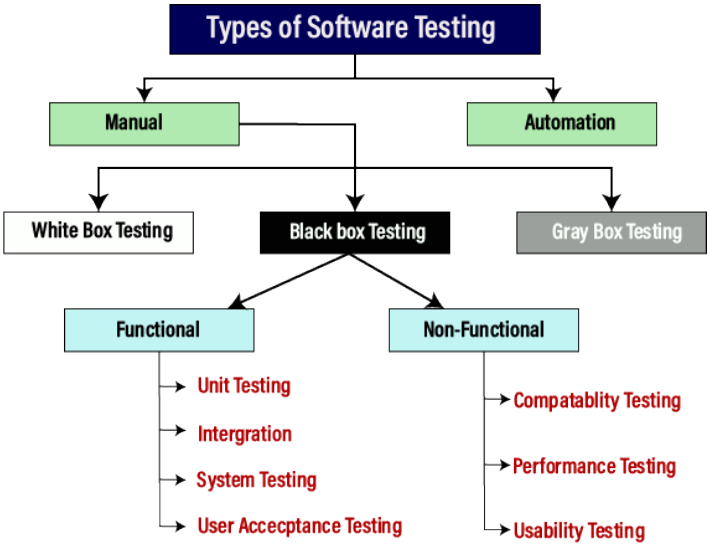

Figure 1: Different Types of Software Testing

testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

3.2 Features of JUnit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

3.3 What is a Unit Test Case ?

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

3.4 JUnit - Environment Setup

JUnit is a framework for Java, so the very first requirement is to have JDK installed in ones' machine. To setup JUnit in a PC, follow the step by step procedures from the site https://www.tutorialspoint.com/junit/junit_environment_setup.htm

3.5 Annotations for Junit Testing

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

- **@Test** annotation specifies that method is the test method.
- **@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).
- **@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.
- **@Before** annotation specifies that method will be invoked before each test.
- **@After** annotation specifies that method will be invoked after each test.
- **@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

3.6 Methods of Assert Class

The common methods of Assert class are as follows:

- **void assertEquals(boolean expected,boolean actual):** checks that two primitives/objects are equal. It is overloaded.
- **void assertTrue(boolean condition):** checks that a condition is true.
- **void assertFalse(boolean condition):** checks that a condition is false.
- **void assertNull(Object obj):** checks that object is null.
- **void assertNotNull(Object obj):** checks that object is not null.

4 Implementation

4.1 Automated Testing: Simple JUnit Example in Eclipse IDE

- **Write the Program Logic:** Let's write the logic to withdraw an amount of money from a bank account using ATM.

```
1  package com.javatpoint.logic;
2  public class BankAccount{
3      public double balance;
4
5      public Deposit(double amount){
6      };
7
8      public static double Withdrawal(double amount){
9          double min_balance = balance - amount; // min_balance >= 500
10         if ((amount<50000) && (min_balance>=500)){
11             balance = balance - amount;
12         }
13         else{
14             return; // Generate Error
15         }
16         return balance;
17     }
18 }
```

- **Write the Test Case:** Here, we are using JUnit 4, so there is no need to inherit TestCase class. The main testing code is written in the testFWithdrawal() method. But we can also perform some task before and after each test, as you can see in the given program (Consider the current balance is 25,000 BDT).

```
1  package com.javatpoint.testcase;
2
3  import static org.junit.Assert.*;
4  import com.javatpoint.logic.*;
5  import org.junit.Test;
6
7  public class TestLogic {
8
9      @Test
10     public void testFWithdrawal(){
11         assertEquals(25000, Calculation.Withdrawal(new double{-500}));
12         assertEquals(23500, Calculation.Withdrawal(new double{1500}));
13         assertEquals(24500, Calculation.Withdrawal(new double{500}));
14         assertEquals(22000, Calculation.Withdrawal(new double{3000}));
15         assertEquals(24000, Calculation.Withdrawal(new double{1000}));
16
17     }
18 }
```

To run this example, right click on TestLogic class -> Run As -> 1Junit Test. Let's see the output displayed in Eclipse IDE (Fig. 2).

As you can see, when we pass the negative values, it throws AssertionError because first time testFWithdrawal() method returns (balance+500) instead of (balance-500). It means our program logic is incorrect.

- **Correct Program Logic:** As you can see, program logic to withdraw money from a bank account is not correct because it doesn't return reduced balance after withdrawal money in case of negative values. The correct program logic is given below:

```
1  package com.javatpoint.logic;
2  public class BankAccount{
3      public double balance;
4
5      public Deposit(double amount){
6      };
7
8      public static double Withdrawal(double amount){
9          double min_balance = balance - amount; // min_balance >= 500
10
11         // Now first condition checks whether the amount is negative or not
12         if ((amount>=500) && (amount<50000) && (min_balance>=500)){
13             balance = balance - amount;
14         }
15         else{
16             return; // Generate Error
17         }
18         return balance;
19     }
20 }
```

If we run the junit program again, you will see the following output (Fig. 3).

4.2 Performance Evaluation

We can evaluate the testing performance by the following formula,

$$TestingPerformance(\%) = \frac{SuccessfulTestCases}{TotalNumberofTestCases} \times 100 \quad (1)$$

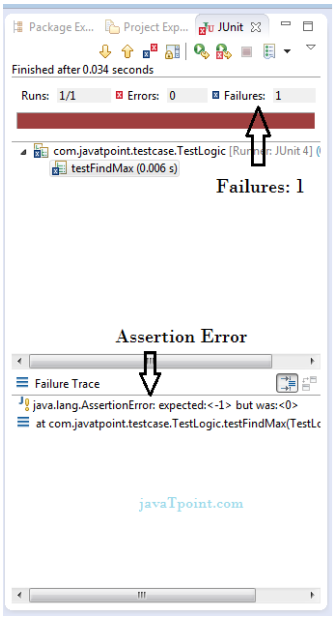

Figure 2: Assertion Error in Eclipse IDE

Figure 3: Error Free JUnit Output in Eclipse IDE

- For **Automated Testing** the Testing Performance is $= (4/5) * 100 = 80\%$

5 Lab Task (Please implement yourself and show the output to the instructor)

- Generate test cases for Balance Checking, Balance Transfer and Request Statement of a Banking ATM System for both Automated Testing.
- Implement the test cases and evaluate performances for Balance Checking, Balance Transfer and Request Statement of a Banking ATM System for Automated Testing.

6 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.