INTRODUCTION TO SOFTWARE ENGINEERING

Lesson 06 – Software Testing and Quality Assurance

OUTLINE

1. Types of testing
2. Test-driven development (TDD)
3. Automated testing tools

1. TYPES OF TESTING

Software testing is the act of checking whether software satisfies expectations.

Software testing is often used to answer the question: Does the software do what it is supposed to do and what it needs to do?

Information learned from software testing may be used to improve the process by which software is developed.

1. TYPES OF TESTING

Unit Test	Test individual component
Functional Test	
Integration Test	Test integrated component
System Test	Test the entire system
Acceptance Test	Test the final system

1.1. UNIT TEST

A testing technique wherein the simple or smallest units of a software program are examined personally for bugs and errors.

These smallest units may be examined independently and personally as they do now no longer want to communicate with every other software program module and are absolutely useful.

Techniques of unit testing:

- **Statement Coverage**: It guarantees that the testing covers all of the statements in a code.
- **Decision Coverage**: Software and any software relies upon decision making system and makes use of conditional statements such as though and else.
- **Branch Coverage**: Generally packages contain branches and employ break statements or goto statements which reason a bounce from one branch of this system to different. Thus it needs to be ensured that every branch of this system is examined. Most of instances it isn't viable to check the entire branches. In such instances, it's far typically suggested to gain minimal 80% branch coverage.
- **Condition Coverage**: Condition testing includes testing every conditional statement in software together with the while, for, and do-while statements.

1.2. FUNCTIONAL TEST

Testing the software program for the useful necessities which are referred to inside the software requirements specification document (SRS). It is a form of black box testing and exams best the functionalities.

- **Smoke Testing:** This form of testing focuses on testing the simple capabilities of the software program and is typically concerned in testing the construct of the software program.
- **Regression Testing:** If bugs are discovered in a software program, then they want to be fixed. In solving those bugs, new bugs can also additionally arise. This testing is accomplished to test if the old bugs had been resolved and if any new bugs have now no longer arisen inside the software program.
- **Integration Testing:** All the units are blended collectively to test if they may be speaking well with every different or we will say that they combine with every different with no error. This form of testing is thought as integration testing (In next slide).
- **User attractiveness testing:** After testing the software program for product, the bugs, it's far examined inside the customer environment. If the customer is glad about the product, then the software program is deployed in all the customer's machines. This testing is referred to as consumer attractiveness testing.

1.3. INTEGRATION TEST

A software program typically incorporates numerous modules and applications that undergo unit testing personally. Once the modules are examined personally, they may be included collectively to test how they interact with every different and test any errors that are termed as Integration Testing.

Approaches to integration testing:

- **Big bang technique**: All of the additives and modules of a software program are included as soon as it is done. This form of testing is best useful for small systems.
- **Bottom-up technique**: The modules at the bottom stage are examined first. Once their testing is complete, they may be included with the modules on the higher layer. The testing keeps till all of the modules are examined and included with the pinnacle stage modules.
- **Top-down technique**: This technique is absolutely contrary to the bottom-up technique. The maximum stage modules are examined first after which included the bottom stage modules step through step.

1.4. SYSTEM TEST

The process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application.

- To detect any irregularity between the units that are integrated. System testing detects defects within both the integrated units and the whole system.
- System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or the context of both.
- It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is performed by a testing team that is independent of the development team and helps to test the quality of the system impartial.
- It has both functional and non-functional testing. System Testing is a black-box testing. System Testing is performed after the integration testing and before the acceptance testing.

1.5. ACCEPTANCE TEST

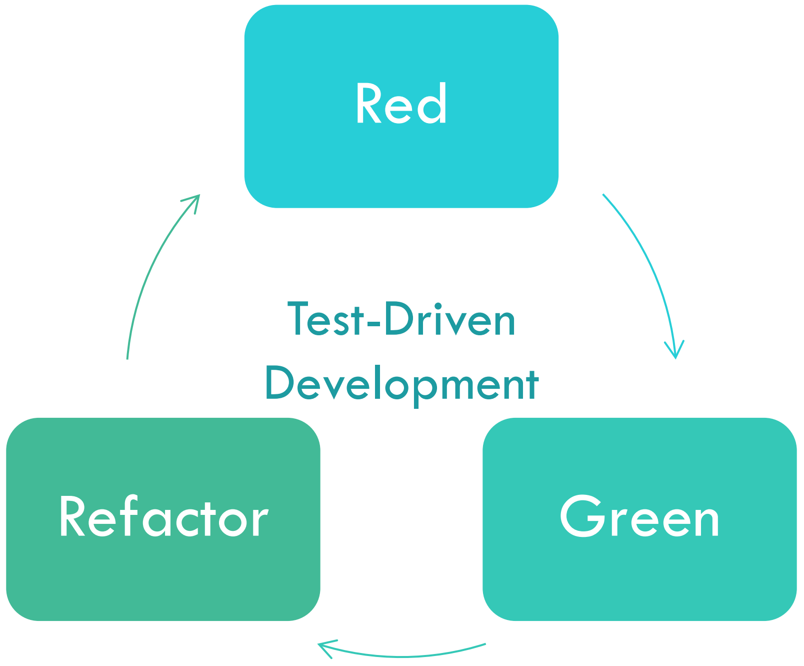
The ultimate level in software program testing earlier than making the software program to be had to be used to the customer. It guarantees that the software program is in keeping with the necessities of the customer and satisfies the customer’s expectations and the necessities referred to within the SRS document.

- **User attractiveness testing:** After testing the software program for all of the bugs, it’s far examined inside the customer environment. If the customer is glad about the product then the software program is deployed in all of the customer machines. This testing is referred to as consumer attractiveness testing.
- **Business attractiveness testing:** This form of testing includes testing whether or not the product or software program meets the enterprise’s purposes, necessities, and goals.
- **Contract attractiveness testing:** It is executed to test if the software program fulfills the necessities referred to inside the agreement which is termed a Service Level Agreement (SLA). There is an exact time restriction wherein the agreement necessities need to be fulfilled as soon as the product is going live.

2. TEST-DRIVEN DEVELOPMENT (TDD)

A software development method that involves writing tests before writing code.

It's an agile technique that helps ensure new features and code improvements are high quality.



2. TDD STEPS

Run all the test cases and make sure that the new test case fails.

Red – Create a test case and make it fail, Run the test cases

Green – Make the test case pass by any means.

Refactor – Change the code to remove duplicate/redundancy.

Repeat the above-mentioned steps again and again

3. AUTOMATED TESTING TOOLS

Automated Testing means using special software for tasks that people usually do when checking and testing a software product. Many software projects use automation testing from start to end, especially in agile and DevOps methods. This means the engineering team runs tests automatically with the help of software tools.

It allows for executing repetitive tasks without the intervention of a Manual Tester.

- It is used to automate the testing tasks that are difficult to perform manually.
- Automation tests can be run at any time of the day as they use scripted sequences to examine the software.
- Automation tests can also enter test data compare the expected result with the actual result and generate detailed test reports.
- The goal of automation tests is to reduce the number of test cases to be executed manually but not to eliminate manual testing.
- It is possible to record the test suit and replay it when required.

3. AUTOMATED TESTING TOOLS

1. End-to-End tests

- End-to-end testing is a type of software testing used to test whether the flow of software from the initial stage to the final stage is behaving as expected. The purpose of end-to-end testing is to identify system dependencies and to make sure that the data integrity is maintained between various system components and systems. End-to-end testing: End-to-end testing, also known as end-to-end functional testing, is a type of testing that validates the flow of a system from start to finish.

2. Unit tests

- Unit testing is automated and is run each time the code is changed to ensure that new code does not break existing functionality. Unit tests are designed to validate the smallest possible unit of code, such as a function or a method, and test it in isolation from the rest of the system.

3. Integration tests

- Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit-tested, integration testing is performed.

3. AUTOMATED TESTING TOOLS

Some testing tools for Java:

- Selenium
- JUnit
- TestNG
- Cucumber
- Appium
- Mockito
- Cypress
- Serenity
- Spock

Some testing tools for C++:

- Google Test
- Boost/test library
- Catch

3.1. JUNIT TEST EXAMPLE

Target class to be tested:

```java
public class Calculation {

    public static int findMax(int arr[]){
        int max=0;
        for(int i=1;i<arr.length;i++){
            if(max<arr[i])
                max=arr[i];
        }
        return max;
    }
}
```

3.1. JUNIT TEST EXAMPLE

Test class:

```java
import static org.junit.Assert.*;
import com.javatpoint.logic.*;
import org.junit.Test;

public class TestLogic {

    @Test
    public void testFindMax(){
        assertEquals(4,Calculation.findMax(new int[]{1,3,4,2}));
         assertEquals(-1,Calculation.findMax(new int[]{-12,-1,-3,-4,-2}));
     }
}
```

3.2. JUNIT TEST EXAMPLE 2

Target class to be tested:

```
import java.util.StringTokenizer;
public class Calculation {
    public static int findMax(int arr[]){
        int max=0;
        for(int i=1;i<arr.length;i++){
            if(max<arr[i]) max=arr[i];
        }
        return max;
    }
    public static int cube(int n){
        return n*n*n;
    }
    public static String reverseWord(String str){
        StringBuilder result=new StringBuilder();
        StringTokenizer tokenizer=new StringTokenizer(str," ");

        while(tokenizer.hasMoreTokens()){
        StringBuilder sb=new StringBuilder();
        sb.append(tokenizer.nextToken());
        sb.reverse();

        result.append(sb);
        result.append(" ");
        }
        return result.toString();
    }
}
```

3.2. JUNIT TEST EXAMPLE 2

```
import static org.junit.Assert.assertEquals;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import com.javatpoint.logic.Calculation;
public class TestCase2 {
  @BeforeClass
  public static void setUpBeforeClass() throws
Exception {
      System.out.println("before class");
  }
  @Before
  public void setUp() throws Exception {
      System.out.println("before");
  }
  @Test
  public void testFindMax(){
    System.out.println("test case find max");
    assertEquals(4,Calculation.findMax(new
int[]{1,3,4,2}));
    assertEquals(-2,Calculation.findMax(new
int[]{-12,-3,-4,-2}));
  }
  @Test
  public void testCube(){
    System.out.println("test case cube");
    assertEquals(27,Calculation.cube(3));
  }
  @Test
  public void testReverseWord(){
    System.out.println("test case reverse word");
    assertEquals("ym eman si
nahk",Calculation.reverseWord("my name is khan");
  }
  @After
  public void tearDown() throws Exception {
      System.out.println("after");
  }
  @AfterClass
  public static void tearDownAfterClass() throws
Exception {
        System.out.println("after class");
    }
}
```

REFERENCES

https://www.geeksforgeeks.org/automation-testing-software-testing/

https://www.geeksforgeeks.org/test-driven-development-tdd/

https://www.geeksforgeeks.org/system-testing/

https://www.javatpoint.com/junit-tutorial

https://en.wikipedia.org/wiki/Test-driven_development

“Design patterns elements of reusable object -oriented software” by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995.