Unit 3 Testing Tools-1

3.1 Testing tools for White box testing

If we go by the definition, “White box testing” (also known as clear, glass box or structural testing) is a testing technique which evaluates the code and the internal structure of a program.

White box testing involves looking at the structure of the code. When you know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification. And all internal components have been adequately exercised.

3.1.1 Testing tools for code coverage:

Code Coverage is a measurement of how many lines, statements, or blocks of your code are tested using your suite of automated tests.

Code coverage shows you how much of your application is not covered by automated tests and is therefore vulnerable to defects. Code coverage is typically measured in percentage values – the closer to 100%, the better. And when you’re trying to demonstrate test coverage to your higher-ups, code coverage tools (and other tools of the trade, of course) come in quite useful.

Over the years, many tools, both open source, and commercial, have been created to serve the code coverage needs of any software development project. Whether you’re a single developer working on a side project at home, or an enterprise with a large team, or working on QA for a start-up, there’s a code coverage tool to suit every need. Code coverage tools are available for many programming languages and as part of many popular QA tools.

1) **Cobertura:** Cobertura is one of the popular open source code coverage tools. It allows you to execute tasks via Maven and Ant, or the Cobertura CLI. You can embed with other multiple QA tools.

   Features:

   - Allows you to measure coverage without having the source code
   - It is one of the best java code coverage tools which helps you to find which parts of your Java program are lacking test coverage
   - Allows you to represent reports in HTML and XML format
   - Helps you to test lines and branches of the class & method

2) **Coverage.py:**
   Coverage.py is another useful code coverage tool. It is one of the best test coverage tools which helps you to monitor Python programs, notes which are parts of the code have been executed.

   Features:

   - Coverage.py helps you to specify what source files you want it to analyze via the configuration file
   - It also helps you to analyze the source to find out code that which could have been executed but was not.

3) **JaCoCo:**

JaCoCo is a free Java code coverage tool distributed under the Eclipse Public License. It is an open source free code coverage tools for Java, which has been made by the EclEmma.

Features:

- JaCoCo offers instructions, line and branch coverage
- It is one of the best java code coverage tools which supports Java 7 and Java 8

- Helps you to test lines and branches of the class & method
- Offers easy to navigate HTML or XML report

4) OpenClover:

OpenClover tool helps you to measures code coverage for Java and Groovy and collects over 20 code metrics. It helps you to display untested areas of your application. It is one of the best test coverage tools that helps you to combine coverage and metrics to find the riskiest code.

Features:

- Helps you to runs your test faster
- Allows you to focus what's necessary for your test
- Both branch and statement coverage support
- Allows you to generate XML-based report that, combined with ReportGenerator, produces TML-based report on coverage
- Helps you to keep the balance between application and tests

5) BullseyeCoverage:

BullseyeCoverage is a code coverage software for C++ code coverage and C that tells you how much of your source code was tested. This tool allows you to perform unit testing, integration testing, and final release.

Features:

- Provides better c++ code coverage measurement
- It is one of the best test coverage tools which helps you to create more reliable code and save time
- Allows you to Include or exclude any portion of the project code
- Merge results from distributed testing.

6) NCover:

NCover is one of the advanced level code coverage tools for.Net programs and applications. It provides support for statement coverage and branch coverage. This code coverage tool is available on open source and as well as on commercial license.

Features:

- .NET code coverage according to your customized needs
- Helps you to test, track and manage a unified coverage number across entire teams
- Detailed and centralized data about coverage
- It is one of the .net code coverage tools that offer extensive documentation and user support
- It is one of the best c# code coverage tools that helps you to perform manual and coverage tests
- Deliver products to market faster and confidently in agile environments

3.2 Testing tools for Unit Testing

Unit testing is a type of software testing where individual units or components of a software are tested.

The purpose is to validate that each unit of the software code performs as expected.

3.2.1 NUnit:

NUnit is a unit-testing framework for all .Net languages.

Initially ported from JUnit, the current production release, version 3, has been completely rewritten with many new features and support for a wide range of .NET platforms.

NUnit was written entirely in the C# language and fully redesigned to get the advantage of many .Net language features. Like custom attributes and other reflection related capabilities.

Test Fixtures

In NUnit we have Test Fixtures containing Tests. A Test Fixture is the class that contain the tests we want to run.

We typically write one test fixture for each class we want to test.

As a convention we name the test fixture <Class to be tested>Tests.cs. e.g. test fixture for Login.cs would be LoginTests.cs

A test fixture will contain a list of tests, typically one for each method in the class being tested.

There are a few restrictions on a class that is used as a test fixture.

- It must be a publicly exported type.
- It must have not be abstract.
- A non-parameterized fixture must have a default constructor.
- A parameterized fixture must have a constructor that matches the parameters provided.

If any of these restrictions are violated, the class is not runnable as a test and will display as an error in the GUI.

SetUp & TearDown:

A method written within a SetUp attribute will be executed before each test.

A method written within a TearDown attribute will be executed after each test.

Asserts and Exception:

When writing tests it is sometimes useful to check that the correct exceptions are thrown at the expected time.

Consider an example to capture data from a temperature sensor.

A test could be written to check that if the temperature is read before initializing the sensor, an exception of type InvalidOperationException is thrown.

To do this the NUnit.net Assert.Throws method can be used. When using this method the generic type parameter indicates the type of expected exception and the method parameter takes an action that should cause this exception to be thrown.

Features of NUnit:

- It powerfully supports the data-driven tests.
- In this, we can execute the tests parallelly.
- It allows assertions as a static method of the asset class.
- It uses a console runner to load and execute tests.
- NUnit supports various platforms such as mobile, .NET core, and compact framework.

3.2.2 JUnit:

It is another open-source unit testing framework, which was written in Java programing language.

It is mainly used in the development of the test-driven environment. Junit offers the annotation, which helps us to find the test method.

This tool helps us to enhance the efficiency of the developer, which provides the consistency of the development code and reduces the time of the debugging.

It is an open-source testing framework for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The unit test case is a code which ensures that the program logic works as expected.

The org.junit package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

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.

Methods of Assert class:

- void assertEquals(boolean expected,boolean actual): checks that two primitives/objects are equal.
- 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

Test Fixture:

When there are multiple test cases in a JUnit class, there could be a common object or objects used by all the test cases.

In this case, there could be specific functions that might be common throughout all the test cases.

This doesn't mean that the test objects have to be shared by all the test cases. The change to the object made in one test doesn't have to be shared across all the tests.

Test Fixture in JUnit is a set of fixed steps in a code that is used as a precondition and few other sets of steps that are used as post condition for all the tests.

In other words, we are identifying those sets of statements that will repeat for all tests and thereby, try setting a fixed environment for our test methods to run.

The purpose of using Test Fixture is to eliminate the duplication of the common code for all the test cases.

setUp() Method:

There are tests that need initialization of certain objects (string, integer, or ArrayList or any object for that matter).

You may create a method public void setUp() in which you could declare the instance variables for the common objects.

Place this setUp() method under the annotation @Before. With the @Before annotation, the framework will run the method setUp() prior to every test case execution.

tearDown() Method:

If you have allocated external resources in a test, you should remember to free the resources too.

The teardown() method could be added for the clean-up of the objects after the test case execution has been completed.

In a similar fashion as the setUp() method, add a method public void teardown() under @After annotation.

The JUnit framework makes sure that after each test case is run, the method under @After is surely executed.

The objects used up in the test have to be set NULL in the teardown() method so that the garbage from the tests get collected.

JUnit Expected Exception Test: @Test(expected):

JUnit provides the facility to trace the exception and also to check whether the code is throwing expected exception or not.

Junit4 provides an easy and readable way for exception testing, you can use

- Optional parameter (expected) of @test annotation and
- To trace the information, "fail()" can be used

Example

```
@Test(expected=IllegalArgumentException.class)
```

By using "expected" parameter, you can specify the exception name our test may throw.

In above example, you are using "IllegalArgumentException" which will be thrown by the test if a developer uses an argument which is not permitted.

Feature of JUnit:

- It offers the assertions for testing expected results.
- In this tool, we can quickly develop a code that enhances the quality of the code.
- This tool can be structured in the test suites, which have the test cases.
- To run the test, it gives the test runners.
- It will take less time to run the test cases.