COMPARISON ANALYSIS OF TESTNG and JUNIT FRAMEWORKS FOR AUTOMATION WITH JAVA

[1]Oshin, [2]Vineet Chaudhary
[1]M.TECH, [2]M. TECH
[1]Department of Computer Science & Engineering, [2]Department of Computer Science & Engineering
[1]Deenbandhu Chhotu Ram University of Science and Technology (Murthal), Sonepat, INDIA
[2]Maharshi Dayanand University, Rohtak, INDIA

Abstract*:* Junit and TestNG work on principle of test driven development. These frameworks are used because Selenium Webdriver lacks the reporting feature. This paper attempts to compare the Junit and TestNG test frameworks for Java. TestNG proved to be better in terms of diverse annotations, maintain dependency tests, easier parameter handling mechanism, better interactive reporting. However, library code from Junit is mainly used and sometimes replicated to TestNG since JUnit libraries have better organized unit tests. TestNG is useful in case of high level testing whereas JUnit is mainly useful for developers in unit testing.

***Index Terms* - JUnit, TestNG, Framework, TDD, Annotations, Assertions**

I. INTRODUCTION

A software test is a part of software, which executes another part of software. It validates whether the code results in the desired state, executes the desired set of events (behavioural testing) or follows the expected flow. Software unit tests help the developer to check the logic of a chunk of some program to be correct. Software regressions can be recognised by running the tests automatically for the changes introduced in the source code. Having a good automation test coverage of the code allows us to develop more features in our application without having to perform lot of manual testing.

Unit Testing: A unit test is for a small unit of code, for example, a method or a class. A successful unit test runs on removing external dependencies from it, for example, by replacing the dependency with a test implementation result or some mock object created by the testing framework. Unit tests are not suitable for testing component interaction or complex user interfaces. Integration tests need to be developed in this case.

End to End Testing: End-to-end testing is a technique for testing if the flow of an application is behaving as expected from start till finish. The purpose of performing end-to-end testing is used to ensure integrity of data and its components and to identify system dependencies. The whole application is tested for crucial functionalities such as database, interfaces, network, communication with the other applications and systems.

There are several testing frameworks available for Java. The most popular of them are JUnit and TestNG. Basically, to keep the test code separate from the real code, unit tests are created in a separate project or separate source folder. Both these frameworks can be work with Selenium, most widely used tool for test web-app automation. The Testing framework roles include- Defining methodology. Drive Application under test (AUT) by creating mechanism. Executing the tests. Reporting results. It needs to be application independent. It needs to be as generic as possible. For software testing, software tests for the complex and critical parts of our application should be written by testers. If we introduce new features in your application, a good test suite also protects you at the time of regression in existing code. It is, in general, safe to ignore trivial code. For eg., it is typical useless to write tests for get and set methods which simply assign values to the fields. Writing tests for these statements is very time consuming, as testing would be done by the Java virtual machine(JVM). The JVM already facilitates this. If you are developing end user applications, it will be good to assume that a field assignment works in Java automatically. If one starts to develop tests for an existing code, it will be a good practice to start writing test cases for the code in which produced most errors in the past. This way one can focus on the important functionalities of the application.

Very primitive way of unit/integration testing while running the automation scripts, is add name of your test scripts in excel, get its result by running one by one in eclipse IDE or equivalent and update result in excel sheet. This is a complex process and takes a lot of time. We have to use TestNG or JUnit in order to avoid. TestNG and JUnit are test framework. One part of automation is developing automation script, another is running it. Automation scripts run by Junit and TestNG along with preparing the test data, managing and running tests, publishing execution report, clearing the test data.

The contribution of the paper is as under:

- The paper details two major TDD frameworks for automation scripting in Java.
- Further, the paper presents a comparison of the JUNIT with a TESTNG in Selenium WebDriver Java test automation based on various parameters such as parameterization, grouping, dependency, added annotations, parallel testing, static variable declaration.

The rest of the paper is organized as under:
Section II presents details of the review of the testing frameworks by various authors. The Junit and TestNG detailed in Section III and IV respectively. The comparison between TestNG and JUnit based on various parameters is presented in section V. The paper is concluded in section VI. In the last, the references are listed.

II. LITERATURE REVIEW

As stated by Pooja et al in [1] that the Agile is a futuristic testing technique and the JUNIT and TestNG implementation with JAVA for creating the automation regression suite. This approach uses TestNg for integration testing and automating a test suite. The author findings emphasis that agile testing risk can be minimized by automating the suite. The preference for Webdriver with JAVA and reporting with TestNG gives a proof that these tools are open source, have importance in automated regression testing and support defect, change and test management.

Need for reducing the testing time is explained by Deepti et. al [2] while improving the software quality and correctness. To implement the approach the author has used Junit and TestNG frameworks as time reduction-quality improvement model. The model is tested for a good test case(one that finds undiscovered error) and TestNG and Junit responses are recorded. Both the tools are fit on quality improvement and time reduction factors.

JUnit is the most popular of the Java testing frameworks as mentioned by Dinesh et. al. [3].An author focussed on Aspect oriented programming, which states that a software system is nothing but a collection of software concerns. By introducing an aspect and then implementing them as a crosscutting concern using Junit and AspectJ has helped create out the test code with before and after aspects. Junit and AspectJ worked as powerful for testing Java open source applications. A comparison study of both the frameworks was also performed.

We were focussing on the ease of using a GUI framework forgetting the need of having an easy API for developers to allow easy test driven development. A safe and robust system is possible with simplified GUI development. This can be made possible by having a developer-friendly library with simple API using TestNG-Abbot as stated by Alex et. al. [4].

JUnit is an open source library for Java. Assertions are important part of Junit and also of Unit testing in the Agile and Extreme Programming methodologies as stated by Louridas [5].

The features of Java Modelling Language (JML) tool when combined with Junit framework can provide a reasonable test coverage as stated by Daniel et. al.[6]. Another framework called JMLUnitNG, which is similar to TestNG but automatically creates and executes number of cases with just primitive type of data. This helped to avoid most data generation by hand.

JUnit 4 remained one of the important unit testing frameworks. Elliotte [7] gave its initial overview. Its various features and advantages in Java based automation are annotations, test exceptions, timed tests , new assertions and build and configuration supports.

TestNG is a TDD framework that supports various testing techniques like unit, integration, system, UAT. On being used with Selenium Webdriver to create an automation framework, TesTNG has advantages for manual testers to execute scripts smoothly as states by Rishab et. al. [8].

Developers and testers frequently use Java testing libraries. As stated by Ahmed [9], maintaining many testing libraries creates a risk of making them redundant. All libraries are not used equally. Few are quickly adopted than others. Few are quickly upgraded. TestNG and JUnit libraries were studied. JUnit libraries, which had JDBC, were frequently used. TestNG library files were less frequently used as compared to JUnit.

As web based automation is always a demand. Selenium is a powerful tool interacting with browsers, frameworks, and languages to make this possible as stated by Fie [10]. Selenium with JMeter makes a powerful tool for both functional and performance automation thus, influencing the product quality to better.

III. JUNIT

JUNIT is the Unit Testing frameworks in JAVA for Test Driven Development (TDD). Test Driven Development: A particular code segment is developed to verify a specific change rather than covering several layers of interacting changes. It requires writing a code that passes a particular test and covers other previous tests. It requires deep thinking and a reduced pace. A JUnit test is a method contained in a class and is used for testing (unit). Such a class is a *Test class* and a Junit method defined in this class is a Test Method. Its annotation is given as @Test annotation. This method executes the code under test. We can then use various other

annotations like an @A*ssert* method, provided by JUnit or another assert frameworks, to validate an expected result against the actual result. These method calls are called *assertions*.
JUnit comes from xUnit family of unit testing frameworks which are used for executing and developing the unit test cases [3]. For functional or regression automation, Junit is designed for test case writing and execution in Webdriver using Java. It has to validate that the code and functions are working as per the requirement. For testing a piece of code using JUnit, we create a class that extends the class of test case and then the various test methods are applied on it [12]. Annotations are the underlying basis of JUnit. Junit uses them to ensure better structure and readability [3]. They are, basically, syntactic metadata added to the testing code. Fig. 1 shows the interaction of JUnit with the test code.This explains that JUnit is mainly a reporting framework but how it interacts with the test code defines its popularity among developers.

3.1 Testing Private Members

JUnit does not have straightforward method of testing the private methods. On the other hand, it is important to test private methods when it contains an algorithm since it requires more unit testing. Using public methods is avoided in this case because their abstraction level is very high and cannot be targeted by test class easily. In a code there can be some private utility methods which work upon passed arguments. However, it becomes important to test their operations. So, in order to help with unit testing of private methods, the Java Reflection API can be used as added component. The *java.lang* and *java.lang.reflect* packages provide required classes for this API. But this approach has its disadvantages:
First, the test code becomes verbose. This results in a code which hard to maintain and understand. Second, since java reflection involves dynamically resolved types at run time, the associated operations become slow due to no optimization.

3.2 Parameterization

JUnit does not have any provision for testing a method with multiple input values. We have to use the *Parameterized Class*. The runner inside JUnit which is known as Parameterized, which eventually runs the same test case using different types of inputs. The JUnit code for testing a method with multiple inputs and using parameterized class is complicated and is difficult to learn and understand.

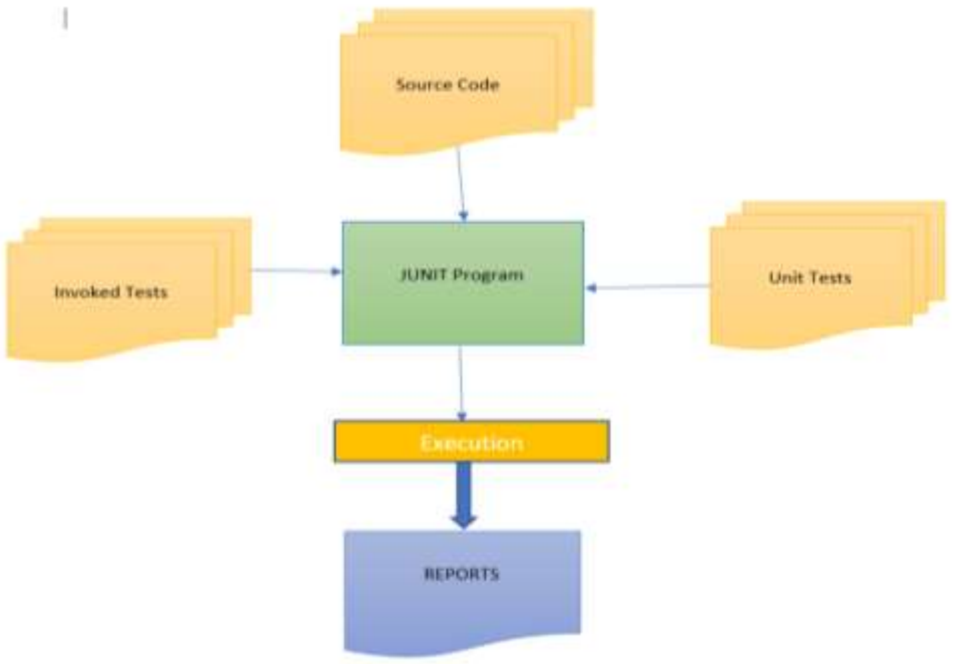

Figure 3.1: Junit Process Model

IV. TESTNG

TestNG (NG means Nest Generation) by *Cedric Beust*, is a testing framework inspired from *JUnit* and *NUnit* .It has introduced new functionalities as compared to Junit to make it more powerful and easier to use. It is an open source automation testing framework. TestNG is similar to JUnit in the basic functions since it is inspired from JUnit but it is much more powerful in few aspects. It is designed in a better way, especially when we are testing integrated classes. Most of the drawbacks of the old frameworks are eliminated by providing the testers and developers an ability to write more powerful test codes having good flexibility using with help of easy grouping, sequencing, annotating and parametrizing. TestNG is another framework similar to JUNIT but it is not the 'de facto' test framework for Java [13]. We can define which test to run before/after. A server is not required for both the frameworks in web application testing. This makes the testing process swift. TestNG is distributed as a JAR file, which must be present in the classpath to execute tests. It integrates with the popular build creation tools: Ant, Maven. Major IDEs and

Java, Scala JVM languages are supported by TestNG. It is friendly with many quality and testing tools (like code coverage tools, mock libraries).

Below is a simple test example:

```
package testTestNG;

import org.testng.annotations.*;

public class ExampleTestNG {

 @BeforeClass
 public void invokeTest() {
//this code is invoked as the test gets instansiated
 {
 @Test(groups = { "fast" })
 public void fastTest() {
   System.out.println("Fast Test");
 }

 @Test(groups = { "slow" })
 public void slowTest() {
   System.out.println("Slow Test");
 }

}
```

The method invokeTest() will be invoked after the test class has been built and before any test method is executed. In this example, if we are running the group fast, fastTest() will be invoked while slowTest() will be skipped and vice versa. The highlights are:

- There is no need to extend a class or implement an interface.
- The annotations that tell TestNG what they are, methods can be called any name.
- A test method can belong to one or more groups.

Once we have compiled test class into the build directory, we can invoke test with the command line, an ant task (below) or an XML file.

Using ANT framework to invoke it:

```
c:> ant
Buildfile: testbuild.xml

test:
[testng] Fast test
[testng] ===============================================
[testng] Suite for Command line test
[testng] Total tests run: 1, Failures: 0, Skips: 0
[testng] ===============================================


BUILD SUCCESSFUL
Total time: 6 seconds
```

Now, we can browse the result at: start testOutcome\index.html

Various TestNG Annotations [2]:

@BeforeSuite: It runs the annotated method before running all tests in this suite.
@AfterSuite: It runs the annotated method after running all tests in this suite.
@BeforeTest: Right before running any test method that belongs to the classes inside the tag, it runs the annotated method.
@AfterTest: Right after running any test method that belongs to the classes inside the tag, it runs the annotated method.
@BeforeGroups: A list of groups before which this configuration method runs. It is guaranteed to run shortly before the first test method belonging to any of these groups gets invoked.
@AfterGroups: A list of groups after which this configuration method runs. It is guaranteed to run shortly after the last test method that belonging to any of these groups gets invoked.
@BeforeClass: It runs before the first test method in the current class gets invoked.
@AfterClass: It runs after all test methods in the current class get invoked.
@BeforeMethod: It runs before each test method.
@AfterMethod: It runs after each test method..
@Test: It runs the main method which ia part of test case.
@BeforeTest: It runs before running any test method belonging to the classes inside the tag.
@AfterTest: It runs after running all the test methods belonging to the classes inside the tag.
@BeforeGroups: It runs before a given list of groups. It runs shortly before the first test method that belongs to any of these groups gets invoked.
@AfterGroups: It runs after a given list of groups. It runs shortly after the first test method that belongs to any of these groups gets invoked.
@BeforeClass: The annotated method will be run before all the test methods in the current class have been run.
@AfterClass: The annotated method will be run after all the test methods in the current class have been run.
@BeforeMethod: The annotated method will be run before each test method.
@AfterMethod: Once each test method, the annotated method will run.
@Test: The annotated method is a part of a test case.

TestNG from Selenium (Java) perspective sums as:
4.1. It gives the ability to produce good HTML Reports of execution
4.2. More options are available in Annotations
4.3. Test cases can be Prioritized more easily
4.4. Test cases can be Grouped easily
4.5. Parallel testing is possible
4.6. Generates better Logs
4.7. Data Parameterization is easily implementable

TestNG implies core unit testing with features like parameterization, dependency testing, suite testing or the grouping concept. It is also useful in high level and complex integration testing where JUnit is not very powerful. It is specially used for testing large test suites [13]. In addition, TestNG also covers core JUnit4 functionality along with its own add-ons. It covers all categories of tests like unit, end-to-end functional, integration and overcomes a few JUnit shortcomings like static method declaration, thus, TestNG's features provide a maintainable intuitive test design. TestNG is having its origin in JUnit and NUnit with new alternate usages that make it more powerful and easier to use, such as:

- Annotations (@ keyword)
- Arbitrarily big thread pools with different policies (all methods in their own thread, one thread per test class) to run the threads
- Independent JDK functions for runtime and logging
- Application server testing by dependent methods
- Tests the code to be multithread safe
- Flexible test configurations
- Support for data-driven testing (using @DataProvider)
- Supports parameters smoothly
- Powerful execution model (no entire TestSuite rerun)
- Supported by a variety of tools and plug-ins (Ant, Maven, Eclipse, IDE)

V. JUNIT VERSUS TESTNG

A developer's decision for switching the framework to TestNG has two major factors. The first is the adaptation of the existing test scripts to the new framework and second is the flexibility of the new framework over JUnit.The migration takes place using the eclipse TestNG pluging refactoring support. The Junit assertions are resolved by importing org.testng.AssertJUnit.*; making the transition smooth. For beginners the difference between the testing frameworks is the purpose it is designed for. Junit works with unit testing. High level testing is done by TestNG. Therefore, TestNG is more aligned to the needs of a functional tester than those of a developer. Below are the capabilities of TestNG that Junit doesn't offer (also in Table 5.1):

Suite Rerun: Another capability of the TestNG that Junit does not have is the capability to rerun failed tests. Its time saving to rerun only failed tests without re-executing the suite is a capable feature.

Dependent Tests: One the TestNG capabilities extensively used in our test cases is the capability to depend test steps on other test step. Failure of one of the dependencies causes the depending ones to be skipped and not as failed. Here, we bind the tests to the preconditions, for example using @BeforeGroup feature. If one of the preconditions fail, the test gets skipped rather than failing.
Testing methodologies performed: Strong code structure is ensured by the Unit tests. End-to-end tests and integrations verify if the product client's requirements are met. TestNG can be used to implement all types of tests, thus, improving quality.
Parameterized Tests: TestNG is a unique framework to allow pass parameters to test methods. Parameters are placed in the testng.xml file making that parameter reusable in the test scripts. This means the time delay can be easily parameterized in the the test scripts using, for example, selenium.WaitForPageToLoad("500") statements. Time delay is adjusted in seconds by just changing the file.
Interactive Reports: TestNG provides an interactive report by default. Designers can use it in printing additional things during the test execution like execution time of last test, taking screenshots at each failed test, or writing test results to Excel.
Parallel Tests Execution TestNG provides support for multi-threading. The parallel execution of such a method is a just by writing a single annotation. TestNG can to execute the same test method multiple times by 'n' number of threads simultaneously.

TABLE 5.1: COMPARISON BETWEEN JUNIT & TESTNG

Comparison Parameters	***Testing Frameworks***	
	JUNIT	**TESTNG**
Test Focus	Unit code testing	High level (integration, system) testing
Parameters configuration	Tough. Using @RunWith and @Parameters where @Parameters method returns a list[] with all the actual values and these are passed to a specific class constructor as arguments	Easy. Setting the testng.xml and defining @DataProvider method
Group Tests	Not Supported	Supported
Dependency Test	Not Supported	Supported using
Annotations excluded		@BeforeTest, @AfterTest, @BeforeSuite, @AfterSuite, @BeforeGroups, @AfterGroups
Prioritization	Not Supported	Supported
Parallel Testing	Not Supported	Supported
Static Declarations	Difficult to handle	No such binding of static declarations for @BeforeClass and @AfterClass

VI. Conclusion

When using selenium, the most important point is selecting the ideal framework. TestNG is similar to JUnit in some aspects like test suite can be created in both these frameworks. Both these frameworks support Timeout tests easily. Some test cases can be ignored in both frameworks. Exception tests of AUT can be created in both the frameworks. Both have some common annotations just with different predefined names. In some cases, JUnit remains as best option but in other cases TestNG can be good. The correct choice can be made only by understanding the framework needs. TestNG is the right tool in moving towards Continuous Deployment and advanced automation features which are trending and emerging. TestNG is useful in making selenium tests easier in generating interesting and easily understandable HTML based reports. The output console in Eclipse or JUnit generates a text-based result whereas the TestNG window is more interactive with a graphical output of the result. Apart from this TestNG cares for a testing scope on a big scale. TestNG already has all the benefits of Junit. It can become a bombshell of reporting and high level testing once integrated with the right tools.

REFERENCES

[1] P. Desai[1], P. Desai[2] and S. Mahale, "*Future of Testing: Agile Testing*", International Journal of Engineering Trends and Technology (IJETT), Vol. 15 Issue 2, 2014.

[2] D. Gauri, R. S. Chillar, "*Implementation of Selenium with JUNIT and Test-Ng*", International Journal of Computer Science and Management Studies (IJCSMS), Vol. 12 Issue 3, 2012.
[3] M. Jain, D. Gopalan ,"*Automated Java Testing: JUnit versus AspectJ*", World Academy of Science, Engineering and Technology (WASET), Vol. 11 Issue 11, 2017.
[4] A. RUIZ_, Y. W. PRICE, "*TEST-DRIVEN GUI DEVELOPMENT WITH TESTNG AND ABBOT*", INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE), VOL. 24, ISSUE 3, 2007.
[5] P. LOURIDAS, "*JUNIT: UNIT TESTING AND COILING IN TANDEM*", INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE), VOL. 22, ISSUE 4, 2005.
[6] D. M. Zimmerman and R. Nagmoti", *JMLUnit: The Next Generation*", International Conference on Formal Verification of Object-Oriented Software (FoVeOOS), pp 183-197, 2010.
[7] E. Harold, *"An early look at JUnit 4"*, Instituto Tecgraf (ITG), 2005.
[8] R. Jain, R. Kaluri,*"Design Of Automation Scripts Execution Application for Selenium Webdriver and TestNG Framework"*, Asian Research Publishing Network Journal of Engineering and Applied Sciences (ARPN JEAS), Vol. 10, No. 6, 2015.
[9] A. Zerouali , "*Analysis And Observations Of The Evolution Of Testing Library Usage*", Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution (SATToSE) , 2017.
[10] F. Wang, W. Du, "*A Test Automation Framework Based on WEB*", 11th International Conference on Computer and Information Science (IEEE/ACIS), 2012.
[11] K. Beck, "*Test-Driven Development: By Example*",JOURNAL OF OBJECT TECHNOLOGY (JOT), pp 216, 2003.
[12] S. K. Swain, S. K. Pani, D. P. Mohapatra,*"Model Based Object-Oriented Software Testing"*, Journal of Theoretical and Applied Information Technology (JTAIT), 2006.
[13] P. Bindal and S. Gupta,"*Test Automation Selenium WebDriver using TestNG*", Journal of Engineering Computers & Applied Sciences(JECAS), Vol. 3, No.9, 2014.