**what is keyword frame work and when do we use it?**

A keyword-driven testing is a scripting technique that uses data files to contain the keywords related to the application being tested. These keywords describe the set of actions that is required to perform a specific step.

A keyword-driven test consists of high and low-level keywords, including the keyword arguments, which is composed to describe the action of a test case.

**what is hybrid framework and when do we use it?**

**“**A hybrid framework is a framework that is created by combining together the features of the different types of QTP frameworks.

**what is POM and modular framework?**

POM has easy test maintenance and reduces the duplication of code. The main advantage of Page Object Model is that if the UI changes for any page, it doesn’t require us to change any tests, we just need to change only the code within the page objects (Only at one place).

Modularity avoids duplicity of code. In future if there is any change in the series of action, all you have to do is to make changes in your main modular method script. No test case will be impacted with the change.

**what is testng and junit?**

JUnit is part of Java and its shipped with most of the IDEs such as Eclipse and NetBeans. There are many testing framework being developed using JUnit.

TestNG is an open source framework. It supports parametrization, data driven testing, parallel execution. JUnit is not much flexible for these.

**Writing unit test cases with testng?**

Writing unit tests with TestNG is much like writing tests with JUnit: you create a class which contains methods annotated with the @Test annotation. Also, you can add methods that are run before or after each test or test case. One feature that is distinct to TestNG is the ability to group tests, making it possible to run only a certain group of tests instead of all[1](https://ace.apache.org/docs/writing-tests.html#fn:1).

All unit tests are placed in the same project they are testing, in a separate directory named test. This allows the tests to be compiled and run independently from the remaining code. It is good practice to have the same package structure for your tests and other code.

**what are important elements in testng.xml**

<!ELEMENT parameter ANY>

|  |
| --- |
| <!ELEMENT method-selectors (method-selector\*) > |

|  |
| --- |
| <!ELEMENT method-selector ((selector-class)\*|script) > |

|  |
| --- |
| <!ELEMENT selector-class ANY> |
| <!ELEMENT script ANY> |

**what are different annotations in testng and junit**

In JUnit , we have to declare “@BeforeClass” and “@AfterClass” method as static method. TestNG is more flexible in method declaration, it does not have these constraints.

run before the first test method in the current class is invoked junit(@BeforeClass) and in testing(@BeforeClass).

expected exception in junit(@Test(expected = ArithmeticException.class) and in testing(@Test(expectedExceptions = ArithmeticException.class)

ignore test in junit(@ignore) and in testing(@Test(enbale=false)

run after each test method in junit(@After) and in testing(@AfterMethod)

**What is group and suite and parallel execution in testing?**

TestNG allows us to perform sophisticated groupings of test methods. Using TestNG we can execute only set of groups while excluding another set. This gives us the maximum flexibility in divide tests and doesn't require us to recompile anything if you want to run two different sets of tests back to back. Groups are specified in testng.xml file and can be used either under the or tag. Groups specified in the tag apply to all the tags underneath.

In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution. It also allows flexible configuration of the *tests* to be run. A suite can contain one or more tests and is defined by the <suite> tag.

TestNG allows the tests to run in parallel or multi-threaded mode. This means that based on the test suite configuration, different threads are started simultaneously and the test methods are executed in them. This gives a user a lot of advantages over normal execution, mainly reduction in execution time and ability to verify a multi-threaded code.

**how to decide which test cases needs to be automated**

It is impossible to automate all testing, so it is important to determine what test cases should be automated first.

The benefit of automated testing is linked to how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are ones that are run frequently and require large amounts of data to perform the same action.

can get the most benefit out of your automated testing efforts by automating:

* Repetitive tests that run for multiple builds.
* Tests that tend to cause human error.
* Tests that require multiple data sets.
* Frequently used functionality that introduces high risk conditions.
* Tests that are impossible to perform manually.
* Tests that run on several different hardware or software platforms and configurations.
* Tests that take a lot of effort and time when manual testing.

**what are the steps in automation testing or when do we automation testing?**

Test Automation Feasibility Analysis

Appropriate Tool Selection

Evaluate the suitable framework

Build the Proof of Concept

Develop Automation Framework

Develop Test Script, Execute and Analyze

**What is the use of dataprovider annotation**

The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.

The name of this data provider. If it's not supplied, the name of this data provider will automatically be set to the name of the method.

In the below example we will pass the data from getData() method to data provider. We will send 3 rows and 2 columns ie. we will pass three different usernames and passwords.

**Difference Between JUnit and TestNG**

In TestNG, Parameterized test configuration is very easy while It is very hard to configure Parameterized test in JUnit.

TestNG support group test but it is not supported in JUnit.

TestNG has a feature to configure dependency test. Dependency test configuration for software web application is not possible in JUnit.

TestNG support @BeforeTest, @AfterTest, @BeforeSuite, @AfterSuite, @BeforeGroups, @AfterGroups which are not supported in JUnit.

Test prioritization, Parallel testing is possible in TestNG. It is not supported by JUnit.

View more features of TestNG @ [THIS LINK](http://www.software-testing-tutorials-automation.com/2014/03/introduction-of-testng-unit-testing.html#more) .

**how to generate reports using testng**

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. In simple words TestNG is a tool that help us to organize the tests and help us to produce the test reports. TestNG framework can be used for automation testing with Selenium (web application automation testing tool).

**Generate Reports Using TestNG.**

TestNG library brings a very convenient reporting feature. Once you execute the tests, TestNG generates a test output folder at the root of the project. It combines two kinds of reports.

Detailed Report.

You can find this report in the <*index.html*> file. It combines the detailed information like the errors, test groups, execution time, step-by-step logs and TestNG XML file.

Summary Report.

It is the trimmed version and informs about the test pass/fail/skip count. You can see it from the <***emailable-report.html***> file. It’s an email friendly report which you can embed and share with the stakeholders.