**What is the use of framework in automation testing?**

A framework is considered to be a combination of set protocols, rules, standards and guidelines that can be incorporated or followed as a whole so as to leverage the benefits of the scaffolding provided by the Framework.

**Advantage of Test Automation framework**

* Reusability of code
* Maximum coverage
* Recovery scenario
* Low cost maintenance
* Minimal manual intervention
* Easy Reporting

Ex: selenium jar file is framework and we add this file jar file in our project. We use classes/methods of selenium framework (WebDriver, Webelement classes, get method)

**What are things u consider/keep in framework?**

Create classes/methods to handle exceptions/custom reports, reading data from database, excel file. constructive blend of various guidelines, coding standards, concepts, processes, practices, project hierarchies, modularity, reporting mechanism, test data injections etc. to pillar automation testing. Thus, user can follow these guidelines while automating application to take advantages of various productive results.

**What are the different types of frameworks we have in selenium/qtp?**

Few most popularly used Test Automation Frameworks are:

* Module Based Testing Framework
* Library Architecture Testing Framework
* Data Driven Testing Framework
* Keyword Driven Testing Framework
* Hybrid Testing Framework
* Behavior Driven Development Framework

**What is data driven frame work and when do we use it?**

Data in an application flows across different modules and layers of the application and so we need to think about how to structure our data and pass it to different layers. The layers can be database, api, messaging engines, browser UI and so on. Since we are focused on test data here, we would talk in terms of how to input and output data through an Automation framework and what capabilities do we have to facilitate that movement of data.

**When do we use it?**

In financial and heathcare sector, data is an extremely important piece of IT system, in fact data handling is to be done with extreme care. With respect to Automation, we will need to think about how we can set data (with multiple data sets) on the browser UI through selenium. We have already seen enough number of examples in Test Data section in terms of data-driven capabilities.

There might be situations where we might already have data in an existing format and we would want to re-use that to set values during Automation. That means we might have different data formats. Luckily we already covered dealing with test data in multiple formats again in the Test Data Section. With increasing number of layers in an integrated application, depending on data at each layer might become tricky at times, especially if each layer is managed by a different team and each team has different timelines and priorities. In such situations, we have to rely on data virtualization. The capabilities to parse data in any of the following formats.

* Excel
* XML
* JSON
* YML
* Database

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

A keyword-driven framework is a table-driven testing or action word based testing. It is a software testing method suitable for both manual and automated testing.

n Keyword Driven Testing, you first identify a set of keywords and then associate an action (or function) related to these keywords. Here, every testing action like opening or closing of browser, mouse click, keystrokes, etc. is described by a keyword such as openbrowser, click, Typtext and so on.

* login to “ Amazon” website – Keyword "login" will be used in our automation framework, to the test the login function or action associated with it.
* logout to " Amazon " website— Keyword "logout" will be used in our automation framework, to test the logout function or action associated with it.

In order to create a Keyword driven framework, you need following things

**Excel Sheet**- Identify the keywords and store them in an Excel sheet

**Function Library**- Function library consist of the function for the business flows ( login button for any website).So when test is executed, it will read the keyword from the Excel sheet and call the functions accordingly

**Data Sheets-** Data sheets is used to store the test data that will be used in the application

**Object Repository**- based on your keyword driven framework you can use an object repository

**Test Scripts-** Based on the design of your framework, you can have test scripts for each manual test case or a single driver script.

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

Hybrid Test framework is a concept where we are using the advantage of both Keyword and Data driven framework. This hybrid test automation framework is what most frameworks evolve into over time and multiple projects. Maximum industry uses Keyword Framework in combination of Function decomposition method.

The hybrid framework that you create should suite your requirements or your stakeholder’s requirements. And there can be a lot of factors that influence your decisions. For example –

* you might want to use a particular feature in your hybrid framework but it might not be useful or necessary for the application you are testing. Example – you might want to use excel sheets to store test data. But in case your application takes the same data every time, it makes more sense to hard-code the data in your code itself or maybe taking it as environment variables
* you might have a disliking for certain features and hence you always tend to use the features that you like. For example, I’m more biased towards functions and hence I tend to use them more than say, using action calls
* your client may have a preference towards certain type of framework as they would have seen a similar sort of framework being used previously. So if you can convince the client, then its well and good, else you would have to stick around with what they have suggested (this is something that I have had to deal with in few of my automation projects)

Taking all the above points into consideration, you can safely say that the features that you select to create your hybrid framework may vary vastly from someone else’s framework. And hence, there is no predefined structure of this type of framework.

**What is POM and modular framework?**

**POM**

Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too many duplicated code is used. Duplicated code could be caused by duplicated functionality and this will result in duplicated usage of locators. The disadvantage of duplicated code is that the project is less maintainable. If some locator will change, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. Beside of that it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts.

**Modular Driven Framework**

In most of the web application we have few set of actions which are always executed in the series of actions. Rather than writing those actions again and again in our test, we can club those actions in to a method and then calling that method in our test script. Modularity avoids duplicacy 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?**

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

* Annotations.
* Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
* Test that your code is multithread safe.
* Flexible test configuration.
* Support for data-driven testing (with @DataProvider).
* Support for parameters.
* Powerful execution model (no more TestSuite).
* Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
* Embeds BeanShell for further flexibility.
* Default JDK functions for runtime and logging (no dependencies).
* Dependent methods for application server testing.
* TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc...

**Example:**

package example1;

import org.testng.annotations.\*;

public class SimpleTest {

@BeforeClass

public void setUp() {

// code that will be invoked when this test is instantiated

}

@Test(groups = { "fast" })

public void aFastTest() {

System.out.println("Fast test");

}

@Test(groups = { "slow" })

public void aSlowTest() {

System.out.println("Slow test");

}

}

**Junit:**

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

**Example to find maximum number for an array**

package com.javatpoint.logic;

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;

}

}

**Writing unit test cases with Testng**

package com.asjava;

import org.testng.annotations.\*;

public class TestNGSimpleTest {

int testInt;

@BeforeMethod

public void setUp() {

testInt = 0;

}

@Test

public void addTest() {

testInt++;

assert (testInt == 1);

System.out.println("add test");

}

@Test

public void subtractTest() {

testInt--;

assert (testInt == -1);

System.out.println("subtract test");

}

}

package com.javarticles.testng;

import org.testng.annotations.Test;

public class UnitLevelTesting {

@Test

public void unitTest1() {

System.out.println("Unit test1");

}

@Test

public void unitTest2() {

System.out.println("Unit test2");

}

@Test(groups="someFeature")

public void unitTest3() {

System.out.println("Unit test3");

}

}

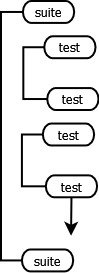
**What are important elements in testng.xml?**

testng.xml is the main configuration file that defines the suite and tests. suite is top level element in TestNG configuration file and is defined by one XML file.

If you need to have more suites then you need to define separate testng.xml file for each suite like database\_testng.xml, feature\_testng.xml, performance\_testng.xml etc.

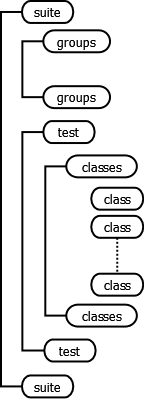
**Structure of testng.xml**

suite is the first element of testng.xml. A suite contains one or more test elements.



Top level structure of testng

A test is made of one or more classes and a class is made or one or more methods.



TestNG configuration

**What are different annotations in testng and junit?**

**TESTNG:**

* **@BeforeSuite** Annotates methods that will be run before any method in a given is run.
* **@BeforeGroups** Annotates methods that will be run before the first method in any of the specified groups is run.
* **@BeforeClass** Annotates methods that will be run before the first method on the current test class is run.
* **@BeforeTest** Annotates methods that will be run before any method in a given is run.
* **@BeforeMethod** Annotates methods that will be run before each test method.
* **@AfterMethod** Annotates methods that will be run after every test method.
* **@AfterTest** Annotates methods that will be run after all the test methods in a given have been run.
* **@AfterClass** Annotates methods that will be run after the last test method on the current class is run.
* **@AfterGroups** Annotates methods that will be run after the last test method belonging to the groups specified in its value attribute has been run. The annotated method is automatically put into these specified groups.
* **@AfterSuite** Annotates methods that will be run after all the test methods in a given have been run.
* **@Factory** When you wants to execute specific group of test cases with different values, you need to use @Factory annotation. An array of class objects is returned by @Factory annotated method and those TestNG will those objects as test classes.
* **@Listeners** are used to with test class. It is helpful for logging purpose.

package automationFramework;

import org.testng.annotations.AfterClass;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class Sequencing {

@Test

public void testCase1() {

System.out.println("This is the Test Case 1");

}

@Test

public void testCase2() {

System.out.println("This is the Test Case 2");

}

@BeforeMethod

public void beforeMethod() {

System.out.println("This will execute before every Method");

}

@AfterMethod

public void afterMethod() {

System.out.println("This will execute after every Method");

}

@BeforeClass

public void beforeClass() {

System.out.println("This will execute before the Class");

}

@AfterClass

public void afterClass() {

System.out.println("This will execute after the Class");

}

@BeforeTest

public void beforeTest() {

System.out.println("This will execute before the Test");

}

@AfterTest

public void afterTest() {

System.out.println("This will execute after the Test");

}

@BeforeSuite

public void beforeSuite() {

System.out.println("This will execute before the Test Suite");

}

@AfterSuite

public void afterSuite() {

System.out.println("This will execute after the Test Suite");

}

}

**JUNIT:**

* **After:** If you allocate external resources in a Before method you need to release them after the test runs.
* **AfterClass:** If you allocate expensive external resources in a BeforeClass method you need to release them after all the tests in the class have run.
* **Before:** When writing tests, it is common to find that several tests need similar objects created before they can run.
* **BeforeClass:** Sometimes several tests need to share computationally expensive setup (like logging into a database).
* **Ignore** : Sometimes you want to temporarily disable a test or a group of tests.
* **Test**: The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.

**What is group and suite and parallel execution in testng**

Group test is a new innovative feature in TestNG, which doesn’t exist in JUnit framework. It permits you to dispatch methods into proper portions and perform sophisticated groupings of test methods. Not only can you declare those methods that belong to groups, but you can also specify groups that contain other groups. Then, TestNG can be invoked and asked to include a certain set of groups (or regular expressions), while excluding another set. Group tests provide maximum flexibility in how you partition your tests and doesn't require you to recompile anything if you want to run two different sets of tests back to back.

Groups are specified in your testng.xml file using the <groups> tag. It can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.

A test suite is a collection of test cases intended to test a behavior or a set of behaviors of software program. 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.

<suite> is the root tag of your testng.xml. It describes a test suite, which in turn is made of several <test> sections.

**How to decide which test cases needs to be automated?**

Step 1:

Identify the parameters on which you will base your test case as a candidate for automation.

As of now I am identifying the below parameters, you can have your own parameters depending on your application.

* Test case executed with different set of data
* Test case executed with different browser
* Test case executed with different environment
* Test case executed with complex business logic
* Test case executed with different set of users
* Test case Involves large amount of data
* Test case has any dependency
* Test case requires Special data

Break each application into modules. For each module, analyze and try to identify the test cases which should be automated based on the parameters.

Consolidate and group the number of test cases for each module

We should also take into account the below attributes which forms the basis of deterring the ROI:

* Purchasing and licensing cost of the tool
* Time to develop the scripts
* Time to maintain the scripts.
* Time to analyze the results manually and automatically
* Time and cost to train the resources.
* Management overheads

**Difference between junit and testing?**

Similarities Between JUnit and TestNG

* We can create test suite in JUnit and TestNG both frameworks.
* Timeout Test Is possible very easily in both the frameworks.
* We can ignore specific test case execution of software web application from suite in both the frameworks.
* It is possible to create expected exception test for software web application in both the frameworks.
* Annotations - Few annotations are similar in both frameworks suite like @Test, @BeforeClass, @AfterClass. JUnit's Annotations @Before and @After are similar to TestNG's @BeforeMethod and @AfterMethod annotations.

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.

**How to generate reports using testing?**

TestNG, by default, generates a different type of report for its test execution. This includes an HTML and an XML report output. TestNG also allows its users to write their own reporter and use it with TestNG. There is also an option to write your own loggers, which are notified at runtime by TestNG.

There are two ways to generate a report with TestNG:

**Listeners:** For implementing a listener class, the class has to implement the org.testng.ITestListener interface. These classes are notified at runtime by TestNG when the test starts, finishes, fails, skips, or passes.

**Reporters:** For implementing a reporting class, the class has to implement an org.testng.IReporter interface. These classes are called when the whole suite run ends. The object containing the information of the whole test run is passed to this class when called.

Here, we will have four different examples to demonstrate four different cases of reporting and logging:

Custom Logging: This example illustrates how to write your own logger.

Custom Reporter: This example illustrates how to write your own reporter.

HTML and XML report : This example illustrates the default HTML and XML report generated by TestNG.

JUnit Reports : This example illustrates how to generate JUnit reports from TestNG reports.