**How to generate reports using 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.

In this chapter, 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.

**Difference between junit and testing?**

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.

**what is the use of dataprovider annotation?**

The annotated method must return an Object[] where each Object[] can be assigned the parameter list of the test method.

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.

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

Here are the steps that we follow before taking up test automation

**Step 1 -** Why to automate? Understanding the purpose of automation and what key stakeholders are expecting from test automation helps in making key decisions related to tool and test cases selection. Some automate to save manual testing cost, some automate to reduce manual testing time (reduce time to market), some automate to improve accuracy, etc.

**Step 2 -** When to automate? Ensure timing of automation is right. At minimum- product should be stable, and manual testing should be in place.

**Step 3 -** What to automation and what not to automate? Test selection criterion. Based on the objective identified under step 1, one needs to select test cases. For e.g. If achieving higher ROI is the objective, then one may select simple and less complex test cases that are frequently executed in the regression cycle. If reducing manual testing time is the objective, then one may automate whatever is possible whether simple or complex.

**Step 4-** Which tool to use? Here one needs to do tool compatibility and select the best tool that works for the product and the organization.

**Step 5-** How to automate? At this stage, one has to focus on automation design and framework for the product.

**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

* **Step 2:** Break each application into modules. For each module, analyze and try to identify the test cases which should be automated based on the parameters. This list will vary for projects to projects and can also be enhanced to suite your needs:

Y – Yes

N – No

In a similar way, for all modules, this list can be used to identify the automation candidate test cases.

**Step 3:** Consolidate and group the number of test cases for each module

**Step 4:** Once you have identified all the granular level details, you can present them in the below way. We are now progressing to calculate the ROI.

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

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

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

The different annotations in testng are

**Annotation Description**

@BeforeSuite The annotated method will be run only once before all tests in this suite have run.

@AfterSuite The annotated method will be run only once after all tests in this suite have run.

@BeforeClass The annotated method will be run only once before the first test method in the current class is invoked.0

@AfterClass The annotated method will be run only once after all the test methods in the current class have run.

@BeforeTest The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

@AfterTest The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.

@BeforeGroups The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

@BeforeMethod The annotated method will be run before each test method.

@AfterMethod The annotated method will be run after each test method.

@DataProvider Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method.

@Factory Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ].

@Listeners Defines listeners on a test class.

@Parameters Describes how to pass parameters to a @Test method.

@Test Marks a class or a method as a part of the test.

The different annotations in junit are

**@Test:** The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case. To run the method, JUnit first constructs a fresh instance of the class then invokes the annotated method. Any exceptions thrown by the test will be reported by JUnit as a failure. If no exceptions are thrown, the test is assumed to have succeeded.

**@Test (expected = Exception.class):** Sometimes we need to test the exception to be thrown by the test. @Test annotation provides a parameter called 'expected', declares that a test method should throw an exception. If it doesn't throw an exception or if it throws a different exception than the one declared, the test fails.

**@Test(timeout=100):** Somethimes we need to mesure the performance interms of time. The @Test annotations provides an optional parameter called 'timeout', which causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds).

**@Before:** When writing tests, it is common to find that several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before the Test method. The @Before methods of super classes will be run before those of the current class.

**@After:** If you allocate external resources in a Before method you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method. All @After methods are guaranteed to run even if a Before or Test method throws an exception. The @After methods declared in superclasses will be run after those of the current class.

**@Ignore:** Sometimes you want to temporarily disable a test or a group of tests. Methods annotated with Test that are also annotated with @Ignore will not be executed as tests. Also, you can annotate a class containing test methods with @Ignore and none of the containing tests will be executed. Native JUnit 4 test runners should report the number of ignored tests along with the number of tests that ran and the number of tests that failed.

**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.

Flexible test configuration.

Support for data-driven testing (with @DataProvider).

Support for parameters.

Allows distribution of tests on slave machines.

Powerful execution model (no more TestSuite).

Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).

Dependent methods for application server testing.

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

**what is POM and modular framework?**

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on.

The POM was renamed from project.xml in Maven 1 to pom.xml in Maven 2. Instead of having a maven.xml file that contains the goals that can be executed, the goals or plugins are now configured in the pom.xml. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

**Modular 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 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 the use of framework in automation testing?**

A set of guidelines like coding standards , test-data handling , object repository treatment etc... which when followed during automation scripting produce beneficial outcomes like increase code re-usage , higher portability , reduced script maintenance cost etc. Mind you these are just guidelines and not rules; they are not mandatory and you can still script without following the guidelines. But you will miss out on the advantages of having a Framework.

If we have a group of testers and suppose if each project implements a unique strategy then the time needed for the tester become productive in the new environment will take long.

To handle this we cannot make changes to the automation environment for each new application that comes along. For this purpose we use a testing framework that is application independent and has the capability to expand with the requirements of each application.

Also an organized test framework helps in avoiding duplication of test cases automated across the application.

In short Test frameworks helps teams organize their test suites and in turn help improve the efficiency of testing.

Framework is nothing but it is separate project contains packages/classes/methods. Generally framework contains reusable components/code.

Eg: In framework I have created classes/methods to handle exceptions/custom reports, reading data from database, excel file.

It will be a jar file in java or DLL in C#.

In automation code we add this jar/dll’s (using buildpath->add external jar files option) and call appropriate classes/method Whenever it is required.

Eg: 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)

**Different types of frameworks we have in selenium/Qtp?**

Module Based Testing Framework

Library Architecture Testing Framework

Data Driven Testing Framework

Keyword Driven Testing Framework

Hybrid Testing Framework

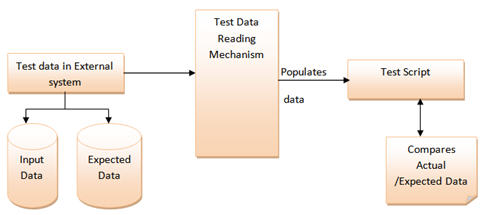
Behavior Driven Development Framework

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

To test same functionality more than once with different set of data/input data. We store the test data in external files (excel/csv) or table.

While automating or testing any application, at times it may be required to test the same functionality multiple times with the different set of input data. Thus, in such cases, we can’t let the test data embedded in the test script. Hence it is advised to retain test data into some external data base outside the test scripts.

Data Driven Testing Framework helps the user segregate the test script logic and the test data from each other. It lets the user store the test data into an external database. The external databases can be property files, xml files, excel files, text files, CSV files, ODBC repositories etc. The data is conventionally stored in “Key-Value” pairs. Thus, the key can be used to access and populate the data within the test scripts.



Data driven testing is where the test input and the expected output results are stored in a separate data file (normally in a tabular format) so that a single driver script can execute all the test cases with multiple sets of data.

The driver script contains navigation through the program, reading of the data files and logging of the test status information.

**Steps involved:** First step involves of creating test data file (test data.csv)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case | Number1 | Operator | Number2 | Expected Result |
| Addition | 10 | + | 10 | 20 |
| Subtraction | 12 | - | 8 | 4 |
| Multiplication | 8 | \* | 5 | 40 |
| Divide | 14 | / | -7 | -2 |

This data file contains the different types of input data which will be given to the driver script.

In the next step we create a driver script and make references to the test data file.

data = open ( ’ testdata.csv’ ) . read ( )

l i n e s = data . s p l i t l i n e s ( ) #excluding the header row

for line in lines:

Read Number1

Read Number2

Read Operator

Calculate the result using the Operator on

Number 1 and Number2

Compare the result to the expected result

This driver script reads the data from the data file computes the value and compares it with the expected result from the data file.

**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.

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.

In 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.

Eg: **login** to "gmail" website – Keyword "login" will be used in our automation framework, to the test the login function or action associated with it.

**logout** from "gmail" website— Keyword "logout" will be used in our automation framework, to test the logout function or action associated with it.

|  |  |
| --- | --- |
| **Keywords** | **Description** |
| Login | Login to gmail site |
| Emails | Check for Emails |
| logouts | Log out from gmail site |
| Notifications | Find unread notifications |

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

1. **Excel Sheet**- Identify the keywords and store them in an Excel sheet
2. **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
3. **Data Sheets**- Data sheets is used to store the test data that will be used in the application
4. **Object Repository**- based on your keyword driven framework you can use an object repository
5. **Test Scripts**- Based on the design of your framework, you can have test scripts for each manual test case or a single driver script

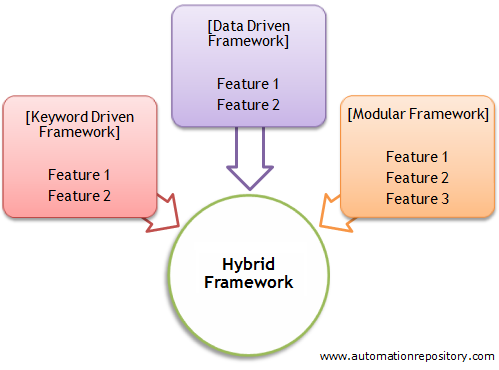
Keyword Driven Testing is done due to following reasons:

1. Common components handled by standard library
2. Using this approach tests can be written in a more abstract manner
3. High degree of reusability
4. The detail of the script is hidden from the users
5. Users don't have to use the scripting languages
6. The test is concise, maintainable and flexible

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

<http://www.automationrepository.com/2013/07/qtp-hybrid-framework-the-basics/>

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



The combination of any two or more frameworks so far is a hybrid framework.

No one framework works effectively for a certain project. Hybrid framework is what gets used most often.

A hybrid framework is a **collection of features** from the other framework types. A hybrid framework is one that **suites your requirements**.

1. what is POM and modular framework?
2. what is testng and junit?
3. writing unit test cases with testing?
4. what are important elements in testng.xml?
5. what are different annotations in testng and junit?
6. what is group and suite and parallel execution in testing?
7. how to decide which test cases needs to be automated?
8. what are the steps in automation testing or when do we automation testing?
9. write manual test cases (if they are not there)?out of all manual test cases we will pick test cases that are of regression testing (test cases that needs to be executed)  put all positive scenarios in regression. we dont put any UI related changes in regression testing?
10. what is the use of dataprovider annotation?
11. Difference between junit and testing?
12. how to generate reports using testng?
13. Creating Jar file and using that JAR file in selenium project?

<http://doc.alertsite.com/synthetic/monitors/selenium/create-runnable-jar-from-selenium-script-using-eclipse.htm>

<https://www.joecolantonio.com/2014/05/29/selenium-create-an-executable-java-test/>

First we need to have selenium installed and added to eclipse.

A project is created and once if it is executed without errors we are set to go further.

On the class file you need to right click and select export option from it. Then you need choose java>runnable file.

Then it will give you an option to select location were you want to save jar file and select the selenium class file from options. Click next and then finish.

Then you will get a dialogue box just click ok irrespective of the dialogue.

Now open command prompt for running that jar file.

Use commands: cd desktop

Cd folder name

Then type java –jar filename.jar

You will be navigated to the page on browser

And you can see a prompt test done in command prompt window if it is executed successfully.