1. what is hybrid framework and when do we use it?

**Hybrid framework is the most commonly used framework in test automation projects.**

A hybrid framework is a collection of features from the other framework types.

A hybrid framework is one that suites your requirements.

Hybrid framework is the combination of both keyboard driven and data driven   
frameworks. About Implementation of the same at the time of creating of frame   
work need to take care of all test objects those are not going to be changed in   
 application should be hardcoded and most frequently changing objects should be   
 taken from the data sheet.

1. what is POM and modular framework

**Page Object Model** is a design pattern that can be implemented using selenium webdriver. It essentially models the pages/screen of the application as objects called Page Objects, all the functions that can be performed in the specific page are encapsulated in the page object of that screen. In this way any change made in the UI will only affect that screens page object class thus abstracting the changes from the test classes.

**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.

1. what is testng and junit?

TestNG is a[Testing](http://www.guru99.com/software-testing.html)framework that overcomes the limitations of another popular testing framework called JUnit. The "NG" means "Next Generation". Most Selenium users use this more than[JUnit](http://www.guru99.com/junit-tutorial.html)because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium.

There are three major advantages of TestNG over JUnit:

Annotations are easier to understand Test cases can be grouped more easily Parallel testing is possible

**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.

1. writing unit test cases with testng

Writing a test in TestNG basically involves the following steps:

Write the business logic of your test and insert TestNG annotations in your code.Add the information about your test (e.g. the class name, the groups you wish to run, etc.) in a testng.xml file or in build.xml. Run TestNG.

1. what is the use of framework in automation testing

The framework introduces high level of modularization which leads to easier and cost efficient maintenance. The framework is pretty much scalable If the changes are implemented in one part of the application, only the test script representing that part of the application needs to be fixed leaving all the other parts untouched. Framework is nothing but it is separate project contains packages/classes/methods. generally framework contains reusable components/code

Ex: in framework i created classes/methods to handle exceptions/custom reports, reading data from database, excel file

It will be a **jar** file in java

It will be **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

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)

1. what are things u consider/keep in framework

Ans:

1. **Test Script modularity:** This concept encourages testers to write code in an object-oriented manner. This allows reusable code to be incorporated throughout any given test. It can also assist in set-up tasks, such as preparing an environment or creating or refreshing database tables.

We use WatiN, NUnit and CruiseControl.NET. We have created a useful wrapper for WatiN (which we call WatiNQA). This wrapper gives us added XPath support that is not native to WatiN. C# gives us our test modularity. The WatiNQA wrapper is separated from the actual test framework to maintain the modularity of the frameworks in use.

1. **Data-driven or keyword-driven testing:** You may have specific scenarios that require an object in a specific state. As a good measure of verification, you can use Microsoft Excel spreadsheets or database queries to gather information to be tested. You can also leverage this to include database verification after a task has been completed.

We query the database to find specific data scenarios. This removes the process of setting up each data scenario through the user interface. After the test is complete, we verify that it is being saved correctly in the database.

1. **Environment to support the test framework:** This would include a continuous build system to launch a test suite at will, on schedule, or even after another event has occurred. With this environment, you can also run different suites of tests, such as smoke tests, functional tests, or browser-based tests. The ability to review test suite results and to view log files are necessary to provide feedback on the success or failure of the test suite. This helps ensure the tests are contributing to the testing efforts. It is also convenient to have a method of linking automated tests with their corresponding manual test cases and any bugs reported on each feature.

Using CruiseControl.NET with NUnit allows us to maintain the test framework with ease. We have it configured to launch several different configurations of test suites, and viewing the results is easy. We’ve also added the convenience of posting the automated test results directly into the test case management tool by creating a NUnit add-in. We use additional test decorators: <code>[Test, TestCaseKey(“”), BugKey(“”)]</code>.

1. **Maintenance friendly:** The framework offers an appropriate solution for the project at hand. It must be easy to use and maintain. By keeping clean code in an organized fashion, it becomes more of a tool than a hindrance. Documentation is always helpful when using the framework.  
     
   We include comments throughout the tests and supporting class files. The objects are configured to leverage reusable code to make writing a test a simple process. We also implement the Façade pattern for our helper methods. This helps us to simplify the tests but still have access to lower-level functions as needed.
2. **The right tools:** Even though many tools exist for automation use, not every tool is adequate for each project. A stable tool that has longevity can provide a standard of excellence on the project. When this happens, the framework is used more and used correctly. Also, the test results have a higher level of trust. This will help avoid automation frameworks to be discarded when changing personnel on the team.

We use C# and WatiN for several reasons. C# is a stable tool that has longevity and great documentation. WatiN is open source and has an active community working to improve it. Though it was originally created for Internet Explorer, a version is available which supports FireFox testing. WatiN deals directly with the Document Object Model (DOM) and can truly represent how the user would truly interact with the Web site.

1. Writing a test in TestNG basically involves the following steps:

Write the business logic of your test and insert TestNG annotations in your code.Add the information about your test (e.g. the class name, the groups you wish to run, etc.) in a testng.xml file or in build.xml.Run TestNG.

11Q) what are different annotations in testng and junit

The annotation supports are implemented in both JUnit 4 and TestNG look similar.

|  |  |  |
| --- | --- | --- |
| Feature | JUnit 4 | TestNG |
| test annotation | @Test | @Test |
| run before all tests in this suite have run | — | @BeforeSuite |
| run after all tests in this suite have run | — | @AfterSuite |
| run before the test | — | @BeforeTest |
| run after the test | — | @AfterTest |
| run before the first test method that belongs to any of these groups is invoked | — | @BeforeGroups |
| run after the last test method that belongs to any of these groups is invoked | — | @AfterGroups |
| run before the first test method in the current class is invoked | @BeforeClass | @BeforeClass |
| run after all the test methods in the current class have been run | @AfterClass | @AfterClass |
| run before each test method | @Before | @BeforeMethod |
| run after each test method | @After | @AfterMethod |
| ignore test | @ignore | @Test(enbale=false) |
| expected exception | @Test(expected = ArithmeticException.class) | @Test(expectedExceptions = ArithmeticException.class) |
| timeout | @Test(timeout = 1000) | @Test(timeout = 1000) |

The main annotation differences between JUnit4 and TestNG are

1. In JUnit 4, we have to declare “@BeforeClass” and “@AfterClass” method as static method. TestNG is more flexible in method declaration, it does not have this constraints.

2. 3 additional setUp/tearDown level: suite and group (@Before/AfterSuite, @Before/AfterTest, @Before/AfterGroup). See more

1. what is group and suite and parallel execution in testing?

Parallelism or multi-threading in software terms is defined as the ability of the software, operating system, or program to execute multiple parts or sub-components of another program simultaneously. 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.

1. how to decide which test cases needs to be automated?

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:

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.

16. Difference between junit and testng?

A) These two frameworks used to perform unit testing of software products have some common feature. First of all, they give testers the possibility to design the unit test cases. Besides that, JUnit and Test NG provide an easy conduction of the timeout test. Both frameworks ensure the creation of expected exception tests. Some annotations for both tools are similar (for example, @AfterClass, @Test, @BeforeClass). Testers are able to ignore the execution of some specific test cases within the suite.

* TestNG supports parallel testing, while JUnit does not have such a feature.
* The process of parameterized test configuration in TestNG is easier than in JUnit (it requires more time and effort).
* JUnit does not support group test, while in TestNG, the specialists are able to administrate certain tests to a large group of people at the same time.
* TestNG supports the dependency tests with the help of such attributes – dependsOnMethods or dependsOnGroup. JUnit does not support this function.
* JUnit is not able to support special annotations that TestNG can. They are @AfterGroups, @AfterSuite, @AfterTest, @BeforeTest, @BeforeSuite, @BeforeGroups.

17. How to generate reports using testng ?

A) 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.

STEPS TO GENERATE REPORT USING TESTNG :

Step 1 : Select The Testng Reporting Interface.

TestNG supplies two interfaces to implement reporting in Selenium Webdriver. You can apply any of these two in your projects.

* Testers commonly use the <ITestListener> Interface.
* Another is the <IReporter> Interface which is least popular.

Step 2 : Generate Report Using The <ITestListner> Interface .

Step 3 : Verify The Report Generation Process.

2- GENERATE REPORTS USING JUNIT

JUnit is another very good tool that can add intuitive reporting feature in your Selenium project. It provides the JUnit <TestWatcher> class to introduce reporting ability.

The JUnit’s TestWatcher class has the <failed()> and <succeeded()> methods which you can override. The JVM would call them automatically whenever it smells a pass or failure.

2.1 Steps To Generate Reports In JUnit Style

We’ve just summarized the summary of the steps that you can use to generate reports using the JUnit plugin.

* Create a new Java class (name it as JUnitTestReporter) that applies the JUnit rules with the help of the TestWatcher() class.
* Override the <succeeded()> method so that the names of passed tests could get displayed at the console with the <Passed> status.
* Override the <failed()> method so that the names of the failed tests could appear at the console with the <Failed> status.
* Create a sample test class as <JUnitSampleTest> which must extend the <JUnitTestReporter> class to utilize the overridden <succeeded()> and <failed()> methods.

Now we’ll explain how simple it is to work with JUnit so that you can quickly generate the summary of the test execution.

2.2- Simple Report Generation Using JUnit.

2.2.1- Generate Reports By Creating A JUnit Test Watcher Class.

Create a simple project in Eclipse and add the below file <JUnitTestReporter.Java> to your project. This file will display the report into the Eclipse console.

2.2.1- Prepare A Sample JUnit Test Suite To Verify Report Generation.

Also, add the below file <JUnitSampleTest.Java> to your project. It’ll be the main test file which you’ll execute from the Eclipse.

2.3- HTML Report Generation Using JUnit.

We’ll create a new JUnit class that would enable the HTML report generation. This class will also override the TestWatcher methods to implement the desired features.

* We’ll define two static members; one is a File object and the second one is BufferWriter’s handle that will help us adding the test execution summary to the report file.
* We’ll use the following JUnit annotations.

@BeforeClass – It’ll help us define the setup() method. It’ll create/open the HTML report file as per the situation.

@AfterClass – We’ll use it for the purpose of cleanup. It’ll also update the HTML report file to add the HTML footer and close all the open handles.

* We’ll override the following two methods.

<succeeded()> method – It’ll write the names and status of the test cases passed during execution.

<failed()> method – It’ll log the names and status of the test cases failed during execution.

* Now, you’ll find the source code of the <JUnitHTMLReporter> class.

3- Generate Reports Using Extent Library.

The third and the last technique for generating some awesome reports is by using the <Extent Report> library. It comes with a rich set of features.

* Ability to generate dynamic HTML logs.
* Represents test case status with the help of PIE Charts.
* Generates step-by-step test case summary.
* Ability to filter reports based on test status.
* It maintains execution history.
* It captures details like OS, Memory, Java version and so on.
* You can attach error screenshots within the report.

Once you have the reporting library, then follow the below steps to use them in a Selenium Webdriver project.

* Create or open a demo project in Eclipse.
* Add the Jar files shown in the above picture as external libraries to your project.
* Also, make sure to add the Jars that are under the <lib> folder else the exception may occur at run-time.