**Test Planning**

1. Determining the scope, objectives and risks of testing
2. Defining the overall approach of testing (test level, entry/exit)
3. Integrating and coordinating the best activities into the software lifecycle activities
4. Making decisions about what to test, the people and other resources required
5. Scheduling the test activities
6. Selecting metrics for test monitoring and control
7. Budgeting for the test activities.
8. **Entry Criteria (Definition of Ready)**

* Availability of testable requirements, user stories, and/or models
* Availability of test items that have met the exit criteria for any prior test levels
* Availability of test environment
* Availability of necessary test tools
* Availability of test data and other necessary resources.

1. **Exit Criteria (Definition of DONE) –** when a test activity is completed, and it should be stopped.
2. Planned tests have been executed
3. A defined level of coverage (e.g. of requirements, user stories, acceptance criteria, risks, code) has been achieved
4. The number of unresolved defects is within an agreed limit
5. The number of estimated remaining defects is sufficiently low
6. The evaluated levels of quality characteristics (reliability, performance, efficiency, and …) are sufficient.

**Test Execution Schedule**

After the various test cases and test procedures are produced, then grouped in test suites, need to then arrange in a test execution schedule that define the order they run.

•Prioritization,

•Dependencies,

•Confirmation tests,

•Regression tests, and

•The most efficient sequence for executing the tests.

**Test Estimation Techniques**

1. **Metric Based** - Estimating the test effort based on **metrics of former similar projects**, or based on typical values

•# of test condictiones

•# of test cases written

•# of test cases executed

•# of defects found

•Time taken to develop test cases

•Time taken to run test cases

1. **Expert Based -** Estimating the test effort based on the experience of the owner(s) of

the testing tasks or by experts

**Test Reports:** Test reporting is about summarizing and communication test activity information

To project stakeholders, both during and at the end of a test activity (e.g., a test level).

1. **Test Progress Report** – During a test activity.
2. **Test Summary Report** – At the end of a test activity.

**Test Report**

1. Summary

2. Information on what occurred

2. Deviations form plan

4. Exit Criteria Status

5. Blocker factors

6. Metrics

7. Residual Risks

8. Reusable test work products

**Test Progress Report**

1. Status of test activities

2. Factors impeding progress

3. Planned testing activities

4. Quality of the test object

**Configuration management** is to establish and maintain the integrity of the products of the software or system through the project and product life cycle.

During test planning, configuration management procedures and infrastructure

(tools) should be identified, documented and implemented.

* All test items are uniquely identified, version controlled, tracked for changes, and related to each other
* All items of testware are uniquely identified, version controlled, tracked for changes

**Defect Management** - Bug Reporting tools such as“JIRA”

In static or dynamic, defects can be identified and reported.

**Defect Incident Report**

**Why TestNG?**

You have control of the test cases.

We can pick what to run? Smoke must pass or the entire regression. With TestNG we can do it with one click.

Running testcases in TestNG without void main java

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...

I started TestNG out of frustration for some JUnit deficiencies which I have documented on my weblog [here](https://beust.com/weblog/2004/08/25/testsetup-and-evil-static-methods/) and [here](https://beust.com/weblog/2004/02/08/junit-pain/) Reading these entries might give you a better idea of the goal I am trying to achieve with TestNG.  You can also check out a quick [overview of the main features](https://www.beust.com/weblog/archives/000176.html) and an [article](https://beust.com/weblog/2004/08/18/using-annotation-inheritance-for-testing/) describing a very concrete example where the combined use of several TestNG's features provides for a very intuitive and maintainable testing design.

Installation:

**Eclipse plug-in**

**Java 1.7+ is required** for running the TestNG for Eclipse plugin.

**Eclipse 4.2 and above is required**. Eclipse 3.x is NOT supported any more, please update your Eclipse to 4.2 or above.

You can use either the [Eclipse Marketplace](https://marketplace.eclipse.org/content/testng-eclipse) or the update site:

**Install via Eclipse Marketplace**

Go to the [TestNG page on the Eclipse Market Place](https://marketplace.eclipse.org/content/testng-eclipse) and drag the icon called "Install" onto your workspace.

**Install from update site**

* Select*Help / Install New Software...*
* Enter the update site URL in "Work with:" field:
  + Update site for release: <https://dl.bintray.com/testng-team/testng-eclipse-release/>.
* Make sure the check box next to URL is checked and click *Next*.
* Eclipse will then guide you through the process.

You can also install older versions of the plug-ins [here](https://beust.com/eclipse-old). Note that the URL's on this page are update sites as well, **not** direct download links.

We are using TestNG libraries instead of java compiler to compile and run our test cases. That is why we don’t have main method in our TestNG classes. TestNG still need java but in the backend not front etnd.

* TestNG gives us neat output. Good reports will be generated and very detailed.

**Importance of TestNG xml file**

Configurations happen here.

**Creating testing.xml file**

Right click on the project + TestNG + Covert to TestNG

**Hierarchy of .xml file**

Test Suite -> Test Folder (Test name) -> Test cases

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Home Page">

<test thread-count="5" name="Test">

<classes>

<class name="classesPackage.Parent"/>

<class name="classesPackage.A"/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

We can play here with .xml file in TestNG

**Include and Exclude Mechanism**

**Only run some test cases?**

We can comment the testing.xml file some classes if we don’t want to run them.

**Don’t run some methods?** We will use the method and exclude keywords as below:

<classes>

<class name="classesPackage.Parent"/>

<class name="classesPackage.Login"/>

**<methods>**

**<exclude name = "MobileLoginHome"/>**

**</methods>**

<class name="classesPackage.A"/>

</classes>

**Only run some methods or one method from a class?** Use keyword “include”

<classes>

<class name="classesPackage.Parent"/>

<class name="classesPackage.Login"/>

<methods>

<exclude name = "MobileLoginHome"/>

</methods>

<class name="classesPackage.A"/>

<methods>

<include name = "OnlyRunMe"/>

</methods>

</classes>

**Note**: We need to have naming conventions in frameworks. For example, all test cases for a feature should start with a similar name such as MobileLoginTitel, MobileLoginUserName, MobileLogInPassword, MobileLoginButton, etc

We can simply, exclude or include all test cases to run that starting with “Mobile” keyword.

<classes>

<class name="classesPackage.Parent"/>

<class name="classesPackage.Login"/>

<methods>

**<exclude name = "Mobile.\*"/>**

</methods>

<class name="classesPackage.A"/>

<methods>

<include name = "OnlyRunMe"/>

</methods>

</classes>

**Groups**: to run only a group of test cases such as smoke, regression, sanity, …

Simply put:

@Test(groups= {"Smoke"})

@Test(groups= {"Sanity"})

@Test(groups= {"Regression"})

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Smoke">

<test name="Smoke Tests">

<groups>

<run>

**<include name="Smoke"/>**

</run>

</groups>

<classes>

<class name="classesPackage.Parent"/>

<class name="classesPackage.Login"/>

<class name="classesPackage.A"/>

</classes>

</test>

</suite>

Or if you don’t want to run smoke, and all others, simple change the include to exclude like:

**<exclude name="Smoke"/>**

If we don’t specify, testNG will run the test cases alphabetically.

If we want to run a test case first and then another one, then we use below:

@Test(dependsOnMethods= {"MobileLoginHome"})

public void MobileLoginOut() {

// Selenium code

System.out.println("Mobile log out ");

}

**dependsOnMethods** is used for showing methods dependencies

**enabled=false: if want to escape a test case because we know there is issue:**

@Test(enabled=false)

public void DisputeSubmit() {

// selenium code

System.out.println("Submitting dispute or claim");}

**TimeOut:** will wait whatever time we specify before failing.

@Test(timeOut = 4000)

public void CrwolinHomePage() {

// selenium code

System.out.println("Wait 40 seconds before failing ...");

}

**Parameterizations:** need to declare in .xml and for each method. To pass data through parameterization**.**

@Parameters({"URL", "APIKey/username"})

@Test

public void RunParameterizaion(String urlName, String key) {

System.out.println( "Is Barn2Door URL openning? " + urlName);

System.out.println("Is the key printing? " + key);

}

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Barn2Door Home Page">

<parameter name="URL" value ="barn2door.com"/>

<parameter name="APIKey/username" value ="123456"/>

<test name="Home Page Testing">

<classes>

<class name="classesPackage.Parameterization"/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**Data Provider Annotation: two ways; from .xml file or parametrize via @DataProvider annotation.**

Global Environment variables can be incorporated through selenium.

Parametrizing with multiple data sets by running tests with multiple combination

**1st way**

<suite name="Barn2Door Home Page">

<parameter name="URL" value ="barn2door.com"/>

<parameter name="APIKey/username" value ="123456"/>

<test name="Home Page Testing">

<classes>

<class name="classesPackage.Parameterization"/>

</classes>

TestNG listeners in Selenium WebDriver With Examples

<https://www.lambdatest.com/blog/testng-listeners-in-selenium-webdriver-with-examples/>

**TestNG Listeners:** such as take screenshot every test case failed.

**Running Tests in Parallel**

<suite name="Smoke" parallel="tests" thread-count="4">

**GroupRunning classes in Parallel**

<test name="Smoke Tests="classes" thread-count="2">