http://www.assertselenium.com/atdd/difference-between-tdd-bdd-atdd/

REST Assured is a Java library for validation of REST web services. It offers a friendly DSL (Domain specific Languages) that describes a connection to an HTTP endpoint and expected results.

**The benefits of REST Assured**

There are three main aspects of REST Assured that make it a powerful library for API testing:

* It removes the need for writing a lot of boilerplate code required to set up an HTTP connection, send a request and receive and parse a response
* It supports a Given/When/Then test notation, which instantly makes your tests human readable
* Since REST Assured is a Java library, integrating it into a continuous integration / continuous delivery setup is a breeze, especially when combined with a Java testing framework such as JUnit or TestNG

**User Acceptance Testing:**

It is done after all types of testing before going to production phase. It will be done by actual users/client to check whether all the business requirements are met. Before doing uat we have to make sure our entry criteria is met

**Entry criteria:**

**Analyze Business requirements -** which we can refer from SRS document, BRD, Project charter, business use cases, process flow diagrams.

**Creation of Test plan –** the approach/strategy in testing the application to make sure that it meets the system requirements by analyzing diff scenarios in terms of actual users. So, for this test cases and test steps are required

**Business use cases serves as input** – test cases

**Prepare test data-** select live dataso that we can get the actual behavior**.** Testers should be familiar with the flow of the database

**Run and record the results-** execute test case and report bugs and retest it again if bugs are fixed

**Implementation:** make sure that the entry criteria is met and the deliverables will be test scenarios, test cases, test results and defect log

**Exit criteria:**

make sure you meet the following to be fixed before moving to production

1. No critical defects open
2. User should satisfy with the flow
3. UAT signoff meeting with stakeholder

**Thinking of UAT tester:**

UAT Tester should possess good knowledge of the business. He should be independent and think as an unknown user to the system. Tester should be Analytical and Lateral thinker and combine all sort of data to make the UAT successful.

**PAGE OBJECT MODEL:**

**Encapsulation** - binding of private variables and public methods

**Page object model-** (we will use the principle of encapsulation here)

1. Every page (Login) in the application have a java class (Login)

in that java class will declare variables as private and methods for that variables as public.

2. Now, we create a test class for that created java class (Login). in that will create object for that java class so that we can access all the public methods which are created for variables from that java class.

**Page Factory:**

In order to support “**Pageobject**” pattern webdriver **supports library** contains a **factory** class. Here we have 2 methods mainly

1. **@Find** – acts like driver.findElement
2. **initElement** – to initialize the webElement of the page object.

**Important feature of PageFactory**:

It contains **cache** which will store all the frequently used webElements.

**Example:**

private WebDriver driver = null;

@FindBy (name="username") // act like driver.findElement(By.id("username"));

private WebElement UN; // store the value in UN

// now we will send the values into UN with the help of this method i.e.., by passing arguments as values to that UN (variable)

public void setUN(String usernamedata) {

UN.sendKeys(usernamedata);

}

**Important:**

In every page you will have a method that will take you to the next page i.e.., return you an object of the next page

Now, we have to create constructor

Public Login (WebDriver driver) //here we have to pass driver as an argument

{ //first we have to create an object for driver then we. can create object

this.driver = driver; // for Login page

PageFactory.initElements(driver, this); // Initialize all the variables in Login page using driver class i.e..,

} // as soon as it is initiated it will find the element by @FindBy.

**In short:**

Each and every page has a java class and when to navigate from page to another then the method which takes you to another page should return the object of the next page.

So, here every page has a java class and a test class. In that test class u have to write the code accordingly and create a object for that classes to access the methods.

**Advantages:**

* Code becomes less and optimized because of the reusable page methods in the POM classes.
* Page Object Patten says operations and flows in the UI (page class) should be separated from verification (test). This concept makes our code cleaner and easy to understand.
* Object Repo is independent of test cases so, we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
* Methods get more realistic names which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

**Interview Questions:**

**Framework:**

**It is a structured way of arranging our**

1. **Scripts**
2. **page library** – it is a method for that particular page
3. **reports and logs**

**Advantages:**

1. We can maintain easy (avoid repetition)

**example:** we can go to page library (method for a particular page) and make the changes so it is very easy to incorporate changes in the application.

1. **Run and Debugging scripts:** everything is in separate folders (screenshots, locators, test base) so that we can debug, run and fix the failed scripts.
2. **Configuration changes:** we can change in the properties file
3. **Easy to understand:** for each and every class we will be creating methods for that variables so that we can refer only the methods whenever and wherever required.

**Creation of Framework:**

**Components of framework:**

1. Crete a maven project and in that we will have

**pom.xml –** Its is an xml file which contains configuration details to build your project that is it helps you to download all the dependencies, jar files, plugins from the google to your project

**It consists of 3 important things –**

**GroupId – it is like a package name example: com.test.automation**

**ArtofactId – it is like a project name example: Framework**

**Version – which version**

**Now add dependencies like**

**Selenium** – it will download all the associated jar files of selenium from the site. Helps to write test case with the help of element locators

**testNG** - execute the selenium script because selenium itself cannot do it

**Log4j** – logging framework (**log** – official record of events)

It is open source and written in java language and can be used with any size of the application. Helps to store the detail flow of selenium test.

**Components of Log4j:**

1. **Loggers:** for recording the flow of information

**Steps to be followed:**

First, create an instance for **logger** class – logger class is a java based utility that has got all generic methods implemented to use Log4j.

Next, define the logger level. They are 5 diff types of levels like

**ALL –** **it will log everything that is it turns all the logs on**

**DEBUG – prints all the debugging information which is more helpful in development stage**

**INFO – prints informational message that highlights the progress of the application**

**ERROR – prints error message that might allow the system to continue**

**FAUTAL – prints system critical information which is causing the application to crash**

**WARN- prints information regarding faulty and unexpected system behaviour**

**OFF -no logging**

1. **Appenders:** they will deliver the recorded events to the destination. Here there are 3 types like

**ConsoleAppender** – they will send to output (console)

**File appender** – they will send to a file

**Rolling file appender** -to a file with maximum size

1. **Layout:** it is responsible for formatting logging information in different styles

**Apache POI** – read and write data form xml file (Excel)

* After that right click on pom.xml - >got to run as and click on maven install -> then all your dependencies will be downloaded from the google.
* Right click on project -> go to maven and update project .so that all dependencies will be updated. That you can see in **maven dependencies.**
* Make sure that in build path you have all jar files and also jdk 1.8

**Structure of the framework:**

**Creating packages:**

**Create in src/main/java**

1. **config –** store url, username, password and other related information
2. **data –** for keeping data sheets
3. **excelReader –** methods and classes which are used in taking data from excel sheet
4. **testbase-** parent class which has all methods that support our scripts
5. **uiActions -** reusable functions that is common methods which we use in interacting pages.
6. **Screenshot-**to capture failed scripts

**Create in src/main/test**

**Homepage**

**Registrationpage**

**Productspage**

**BillingandShipping page**

**Addtocart page**

**Payments page**

**PAGE OBJECT MODEL:**

**Encapsulation** - binding of private variables and public methods

**Page object model-** (we will use the principle of encapsulation here)

1. Every page (Login) in the application have a java class (Login)

in that java class will declare variables as private and methods for that variables as public.

2. Now, we create a test class for that created java class (Login). in that will create object for that java class so that we can access all the public methods which are created for variables from that java class.

**Page Factory:**

In order to support “**Pageobject**” pattern webdriver **supports library** contains a **factory** class. Here we have 2 methods mainly

1. **@Find** – acts like driver.findElement
2. **initElement** – to initialize the webElement of the page object.

**Important feature of PageFactory**:

It contains **cache** which will store all the frequently used webElements.

**Example:**

private WebDriver driver = null;

@FindBy (name="username") // act like driver.findElement(By.id("username"));

private WebElement UN; // store the value in UN

// now we will send the values into UN with the help of this method i.e.., by passing arguments as values to that UN (variable)

public void setUN(String usernamedata) {

UN.sendKeys(usernamedata);

}

**Important:**

In every page you will have a method that will take you to the next page i.e.., return you an object of the next page

Now, we have to create constructor

Public Login (WebDriver driver) //here we have to pass driver as an argument

{ //first we have to create an object for driver then we. can create object

this.driver = driver; // for Login page

PageFactory.initElements(driver, this); // Initialize all the variables in Login page using driver class i.e..,

} // as soon as it is initiated it will find the element by @FindBy.

**In short:**

Each and every page has a java class and when to navigate from page to another then the method which takes you to another page should return the object of the next page.

So, here every page has a java class and a test class. In that test class u have to write the code accordingly and create an object for that classes to access the methods.

**Advantages:**

* Code becomes less and optimized because of the reusable page methods in the POM classes.
* Page Object Patten says operations and flows in the UI (page class) should be separated from verification (test). This concept makes our code cleaner and easy to understand.
* Object Repo is independent of test cases so, we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
* Methods get more realistic names which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

1. Crete project and create package and in that package
2. Below is the sample code for Page Object Model in Selenium:
3. **public** **class** BasePage {
4. **private** By username = By.id("username");
5. **private** By password = By.id("password");
6. **private** By loginBtn = By.name("loginbtn");
7. **public** **void** **userLogin**(String userName, String password) {
8. driver.findElement(username).sendKeys("testuser");
9. driver.findElement(password).sendKeys("testpassword");
10. driver.findElement(loginBtn).click();
11. }
12. }

And the below is the simple code written using Page Factory in Selenium:

1. **public** **class** **BasePage** {
2. @FindBy(id= "username") **private** WebElement userName;
3. @FindBy(id= "password") **private** WebElement password;
4. @FindBy(id= "login") **private** WebElement loginBtn;
5. **public** **void** **userLogin**(String userName, String password) {
6. userName.sendKeys(userName);
7. password.sendKeys(password);
8. loginBtn.click();
9. }
10. }

**Page Object Model:**

pom - a design pattern to create object repo for web ui elements

1. ui should be seperated from verification, so easy to understand

2. object repo is independent of test cases. so, it is resuable and maintained

Testng - ng (next generation)

framework better than junit

1. annotations are easier to understand

2. test cases can be grouped

3. we can do parallel testing by configuring the pom.xml file

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

Flexible test configuration.

Ability to re-execute failed test cases.

**selenium IDE** - having a record and playback options which is present in every automation tool like qtp

**selenium rc -** injects javascript into browsers when the page is loaded

**selenium webdriver** - automation tool to test web apps in diff browsers using diff languages

**selenium grid**- Selenium-Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel.

selenium core-same like selenium web driver but Selenium Core tests run directly in a browser

**emulators** – android

**simulators** - ios

**Http status codes**

200 - OK - success code

201 - Created - Successfully created happened using POST/PUT

400 - Bad Request

401 - Unauthorized-In response of missing token or invalid authentication.

404 - Not Found-This error occurred when resource not found.

500 - Internal Server Error-sually this happens when server is down.

**HP-Quality Center**: (Test management tool) - it is gng to manage all testing operations

we can place our requirements docs, test cases, any defects , execution can also be done here in one common place and shared with team members

So, any member when they perform any action they are logged with all the details- time and day

JIRA - test management tool (bug and issue tracking, project management tool)

API TESTING: BUSINESS LAYER (NOT GUI) - middle layer (giving request to the api and get response from them)

**approach of api testing:**

understand the scope and functionality

apply testing techniques like

a) equivalence classes - placing all test cases into classes. (example no -1 to 1000 - here we can make 4 test cases)

b) boundary value analysis - next level of equivalence classes where test cases are selected at the edges of equivalence classes (stress and negative testing)

c) error guessing (experience comes into picture where mostly guessing will takes place in all kinds of scenarios)

input parameters should be set properly

execute the test cases and compare expected with actual values

**Challenges in API level testing:**

1. testing parameter combinations - for communication will send data values to parameters and those parameters are like requests, here we have to carefully check the possible parameter combinations

2.validating parameters - make sure that data values are within range and data type

3. sequencing the api calls - when multithreading applications it will become a problem because it should follow certain sequence in sending the requests

4. no gui so difficult to give input values

**Custom listeners** - failed test scripts - capturing screenshots through listener classes and attach to testng report

parameterization in test ng - achieved through

1. parameters =

2. @data provider - for multiple suites - parallel testing ->we will supply

attribute and the return type is 2d array object

Parameterization is require to create Data Driven Testing.

TestNG support two kinds of parameterization, using @Parameter+TestNG.xml and using @DataProvider

In @Parameter+TestNG.xml parameters can be placed in **suite level** and **test level**. If

The Same parameter name is declared in both places; test level parameter will get preference over suit level parameter.

using @Parameter+TestNG.xml only one value can be set at a time, but @DataProvider return an 2d array of Object.

If DataProvider is present in the different class then the class where the test method resides, DataProvider should be static method.

There are two parameters supported by DataProvider are Method and ITestContext.

project (10 mem)

4- developers (1 lead) (after RA he will give knowledge transfer)

3- testers (1 lead) (task is in high level document)

RA (requirement analyst) ( he will give knowledge transfer first) - he will ensure that the project is gng acc to agile like iterations (2 weeks period)

tso

Project manager

flow:

RA prepares 1 and 2

1- kick off meeting ( past of project and who all are involved in that)

2- creating hld(high level design document) - this will be mentioned in story card

high level story cards - large applications

low level story cards - small appliations

3-developers will take story cards and work - they will do unit testing and they will build in application whether it is working or not - code review - correct (sign off ) or false (comments and send it back) - local system

development testing ( in real time - server environment)

4-system testing - integration testing - stress testing (more load by giving large amounts of input ex-10,000) -acceptance testing (BA)

5- take production date (gng live)

**TDD** – Mainly, it is done to check the implementation of code. JUnit (unit testing framework) - it is done repeatedly for the source code by developers. and if anything goes wrong then revert back and implement until it becomes perfect. the concept behind this is that whether each unit is functionally working as expected

**Step – 1:** first, we have to create normal classes for the functionalities and it doesn’t need any implementation. Just make sure it runs that’s it.

**Step-2:** think how to test the code bycreate JUnit test cases. It fails because there is no implementation for it.

**Step-3:** now you write the implementation code until it works properly.

So, here testing is done before developing so this is a test-driven approach

**BDD** – Mainly, it is done to check for behavior of the application. (Cucumber)

1. **feature** file in gherkin language, - in the form of statements
2. **step definition file**-implementing the statements into actual code.
3. Then finally, for integrating feature and step definition we have Runner class like Junit runner

**Dry run** gives you a way to quickly scan your features without actually **running** them

**Scenario outline:** values are not hard coded and it will be given in parameters at the end of the scenario outline in the form of table

**ATDD:**

It is a collaborative practice where users, testers, and developers define automated acceptance criteria early in the development process. ATDD helps to ensure that all project members understand precisely what needs to be done and implemented. This workflow usually involves establishing the criteria first, most often from a user perspective, and creating concrete examples. Thereafter, acceptance tests are developed and run to see the results of failure with the right code based on these examples. Minimal code is then developed to run the program, more acceptance tests are run again, and the results are validated. **Refactoring** is then carried out based on the results of the acceptance tests before the final program is developed for use.

**Challenges in POM framework:**

1. As we have seen the above scenario which is quite simple to implement it in most of the scenarios but there would be some complex scenarios which would be really tricky. Let’s take the same example of Gmail login where we simply created a Page method to login “LoginToGmail” which takes the username and password as the argument and return the next Inbox page.

This was a happy scenario, but if we need to test the negative scenario where we are testing that if we pass the invalid credentials then it will keep you on the same login page and will display the “invalid credentials” error. How we will incorporate this behavior in the same method as it always returns the object of the next page.

This problem can easily be handled in the languages like Ruby where the same class method has multiple return types. We will add an extra argument in the method which will tell us whether we are passing the valid or invalid credentials and on the basis of this value we will change the method return type.

This is how we will tackle this problem in Ruby.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def LoginToGmail(username, password, usertype)      @driver.find\_element(username).send\_keys(uname)      @driver.find\_element(password).send\_keys(password)      @driver.find\_element(SUBMIT).submit      if usertype== ‘valid’          return InboxPage.new (@driver)      else'      return GmailLoginPage.new (@driver)      end  end |

If we have to achieve the same thing in languages like Java and C# then we have no other choice other than splitting this method into two separate methods, as these languages do not support different return types for the same class method.  This is how we will do this in Java:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public HomePage LoginToGmailAsValidUser(String username, String password) {      driver.findElement(usernameloc ).sendKeys(username);  driver.findElement(passwordloc  ).sendKeys(password);  driver.findElement(loginButtonloc   ).click();      return new InboxPage(driver)    }    public GmailLoginPage LoginToGmailAsInvalidUser(String username, String password) {      driver.findElement(usernameloc).sendKeys(username);  driver.findElement(passwordloc ).sendKeys(password);  driver.findElement(loginButtonloc   ).click();      return this;    } |

As we can see the first method will only take the valid credentials and return the Inbox page object. The second method will only take the invalid credentials and will return the same GmailLoginPage object.

**Failure testing** is an important part of the manufacturing process, no matter what you are manufacturing. Failure testing is a way to ensure that you are producing a product and service that will not fail under different circumstances and situations of stress, weather, temperature, and so on and so forth

**smoke testing** - done after the software build to verify the critical functionalities of an application

example - gui is responsive, application launches successfull

**integration testing**: testing the modules as a group. it is done after unit and before system and validation

**acceptance testing**: involved in business requirements phase and in the user environment. here business requirements are the input

**Feature testing**: it will add changes to the existing functionality or add new functionality

**Stress Testing**: It checks the application behaviour beyond normal or peak loads. These are related to synchronization issues, memory leaks and race conditions.

**Load Testing**: steadily increase the load on the system till the time it reaches the threshold limit.

listeners - ITestListeners

custom listeners - used to capture screenshot at the run time

onTestFailure

onTestSuccess

onTestSkipped

onTestStart

onTestFinish

jenkins - new job creation (buid time trend and build history and timeline

master - actual jenkins for building jobs - it shouls have a git or svn and maven to build

slaves hear for the master instructions and these slaves will be used to build the testing in diff environments because it is not possib;e to build in one server

slaves work in diff envirnments

slaves will be accessed by master through ssh

git

git fork - making a copy to your remote repo

git clone - making a copy to your local repo

jmeter

right click on test plan -> add thread groups (no of users, loop count) an then give the http path - > execute the test plan

