Web Service Testing Using Open Source Maven

Dr. Bharathi P. T[1], Jagadish K[2]
[1]Assistant Professor, Dept of Master of Computer Applications
[2]Dept of Master of Computer Applications
[1, 2] Siddaganga Institute of Technology, Tumakuru - 572103

Abstract- *The Web service testing is a combination of four open source technologies viz. Arquillian, TestNG, JaCoCo, and Maven. It is aimed at automating server side testing of all the java based products along with generations of code coverage report. The main idea behind any developing a web service is to be able to test server side components which are developed using Java Script. The tests will be such that they will run in the container/application server where the server side component is deployed. The tests will be able to use all the real resources provided by the container instead of mocking them. The next objective is to calculate the amount of code covered by the test cases i.e. code coverage and display it on the code level. Code coverage gives an insight of what has been missed by the test cases and if we are pretty sure that everything is covered by the test cases then one can ask the question, "Why there is still a piece of code not getting under the test radar?". Is that a junk code?*

Keywords- Eclipse, Maven, JaCoCo, Arquillian, TestNG and Wild Fly.

I. INTRODUCTION

The Information Technology (IT) industry is moving towards automating everything it can. The chunk of that automation includes automating the manual testing so that the products can be tested thoroughly, compressively, and as fast as possible. This leads to more frequent software releases with fewer bugs. Also the client provides the feedback early in the processing stage. This helps in changing the software sooner according the clients requirements. The web service focuses on automating the server side tests of the java based products.
The web service is developed using four open source technologies viz. Arquillian, TestNG, JaCoCo, and Maven. Each of the four technologies plays important roles in achieving the overall objective. TestNG is the test framework using which the actual test cases are written. Arquillian helps in running the test cases in the target container e.g. Wild Fly, so that the tests can also use same resources as the real application is using e.g. EJB, CDI, etc. JaCoCo is used in getting the code coverage report. The maven is the glue that binds all the technologies together. Maven works as dependency management and also used as a building tool. Here the authors are using a white box testing in web service, for all the functionalities that are included. It also helps to test the server side components as they are automated. All the products are placed in one place i.e. server side components and testing side components and all the products are deployed in the container, the container will provide the results.

II. METHODOLOGY

In any IT industry, the structured design begins by collecting all the distinguishing inputs and desired yields to make a graphical portrayal. After developing a graphical portrayal, the test information is bolstered through the experiments written in TestNG and each of the parts is combined all around through their characterized duties. Each of the parts work in strong way to achieve the general errand i.e. to convey and run test cases in the holder and get scope report at same time.

A. Existing System

- In the Existing system, the tester has to use a unit testing. In unit testing, the testing is been carried out unit by unit of the product.
- The developers will not be able to check the server side components in the existing system.
- JavaEE components are tested manually, Unit testing of these components is difficult, and these components are to be tested inside the container.

B. Proposed System

It is designed by keeping in mind to eliminate the drawbacks of the existing system. In order to provide the solution for the existing problems, the main focus is on:

- Reducing manual test work.
- Finding code coverage using JaCoCo.
- In-Container testing of JavaEE components.

C. Requirements for Zebra test solution at the server side testing

- The client should have the capacity to compose test cases, convey and keep running the application in the server side.

- The client should have the capacity to produce code scope report in light of the execution of the experiments.
- The Quality Analyst (QA) group should be capable in getting a report saying which of their experiments have passed or fizzled and furthermore the reason of disappointment.

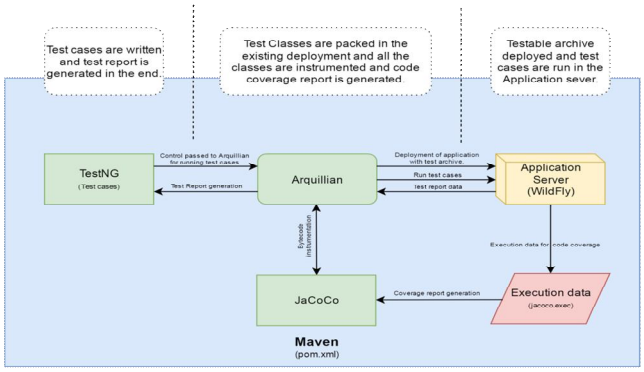


Figure 1: An architectural view of the web service testing including code coverage.

Figure 1 shows an architectural view of the web service testing including the code coverage. This fig1 depicts how the components like Arquillian, TestNG, JaCoCo, and Maven are combined to fulfill the client prerequisites in the testing stage. JaCoCo is used for code scope, Arquillian for sending and running test in the application server side.

The test data is fed through the test cases written in TestNG and each of the components has well defined responsibilities. Each of the components operate in cohesive manner to accomplish the overall task i.e. to deploy and run test cases in the container and get coverage report at same time.

MAVEN:- Maven is a tool developed by Apache with a whole lot of objectives in mind. Here in this proposed project work, the authors have concentrated mainly on two features like dependency management and build management. Firstly, Dependency management helps in downloading all the required libraries/jars for Arquillian, TestNG and JaCoCo and makes them available to the project. Secondly, build management. In case of build management maven has been used to configure:

- test and application classes that need to be built,
- the container adapter (e.g. wild fly-Arquillian-container-remote) to use when Arquillian will try to deploy the application and/or test code in an application container (e.g. Wild Fly), and
- JaCoCo to include/exclude application source files in the code coverage report.

1) Adding a dependency in maven:

Any dependency in maven is defined by three attributes like group id, artifact id, and version. So, to add a dependency one has to know all the three attributes. Let's add a TestNG dependency in our project.

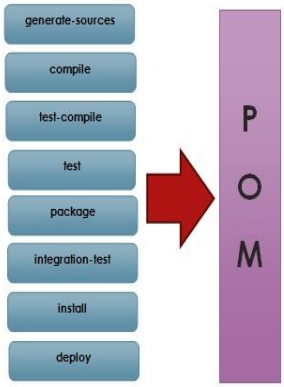


Figure2. Maven lifecycle phases

TestNG: It is a testing framework designed to make simpler a broad range of testing needs, from unit testing to integration testing. Writing TestNG tests include three main steps:

1. Writing the logic for the code to be tested and insert TestNG annotations.
2. Adding information about test case i.e. class name, groups, etc. in testing.xml file.
3. Run the code as TestNG test.

The important annotations used in TestNG are shown in figure 3:

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.
@AfterClass	The annotated method will be run only once after all the test methods in the current class have been run.
@BeforeTest	The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

Figure3. Sample Test annotation of TestNG.

Arquillian: Arquillian is a container-oriented test framework. It picks up where unit tests leave off, targeting the integration of application code inside a real runtime environment.

Arquillian provides a custom test runner for JUnit and TestNG that turns control of the test execution lifecycle from the unit testing framework to Arquillian. From there, Arquillian can delegate to service providers to setup the environment execute the tests inside or against the container.

Figure4. Infrastructure of Arquillian test

JaCoCo: This Java framework calculates code coverage. The coverage report was calculated by JaCoCo method which not only provides ball park view of how much has or has not been covered by the test cases but also stretch a code level view, showing the covered code in green color, partially covered code in yellow color, and missed code in red color.

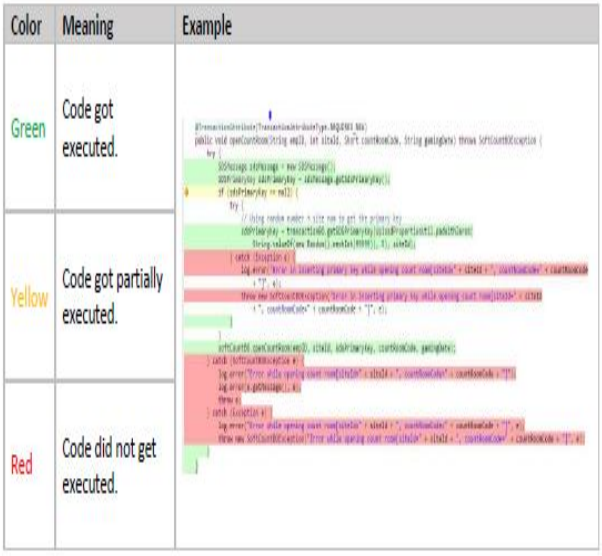
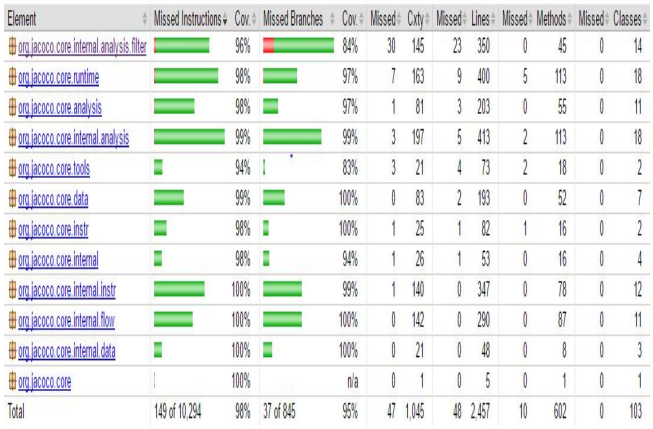


Figure5. Code coverage report calculated by JaCoCo Method.

- Results and discussion



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.core.internal.analysis.filter		96%		84%	30	145	23	350	0	45	0	14
org.jacoco.core.runtime		98%		97%	7	163	9	400	5	113	0	18
org.jacoco.core.analysis		98%		97%	1	81	3	203	0	55	0	11
org.jacoco.core.internal.analysis		99%		99%	3	197	5	413	2	113	0	18
org.jacoco.core.tools		94%		83%	3	21	4	73	2	18	0	2
org.jacoco.core.data		99%		100%	0	83	2	193	0	52	0	7
org.jacoco.core.instr		98%		100%	1	25	1	82	1	16	0	2
org.jacoco.core.internal		98%		94%	1	26	1	53	0	16	0	4
org.jacoco.core.internal.instr		100%		99%	1	140	0	347	0	78	0	12
org.jacoco.core.internal.flow		100%		100%	0	142	0	290	0	87	0	11
org.jacoco.core.internal.data		100%		100%	0	21	0	48	0	8	0	3
org.jacoco.core		100%		n/a	0	1	0	5	0	1	0	1
Total	149 of 10,294	98%	37 of 845	95%	47	1,045	48	2,457	10	602	0	103

Figure 6: Snap shot of Code Coverage image

Figure 6 shows the result of Code coverage. The codes that are green color indicate that the code is covered; yellow color code indicates they are partially covered and red color code indicates that they are missed from JaCoCo method. The image of the Code Coverage has checked the server side testing. It shows the result on how much percentage of the method is checked and if at all any method is missed or not.

III. CONCLUSION

This paper presented the development of Project defines about generating Test cases and code coverage report. It can be used both for developer and the quality analysts. Developers can easily find the junk of code and can easily remove with the help of code coverage and can also easily test real resources in the server side.

REFERENCES

[1] Book: Arquillian Testing Guide by John D. Ament
[2] Book: Next Generation Java Testing: TestNG and Advanced Concepts
[3] Book: Maven: The Definitive Guide by Sonatype Company
[4] Tutorial guides: http://arquillian.org/guides/
[5] JaCoCo official documentation: http://www.eclemma.org/jacoco/trunk/doc/
[6] JaCoCo getting started tutorial: https://www.codeproject.com/articles/832744/getting-started-with-code-coverage-by-jacoco
[7] Maven official documentation: https://maven.apache.org/guides/
[8] Maven tutorial: http://tutorials.jenkov.com/maven/maven-tutorial.html