Volume 6 Issue 1 (2018) 3-5 ISSN 2347 - 3258
International Journal of Advance Research and Innovation

In Container Integration Testing Frame Work

H S Vijaya Kumar*, Vikas S M
Department of Computer Applications, Siddaganga Institute of Technology, Tumakurur, Karnataka, India

Article Info
Article history:
Received 25 January 2018
Received in revised form
20 February 2018
Accepted 28 February 2018
Available online 15 March 2018

Keywords
Integration Testing, Automation Testing, Java Framework, Unit Tests

Abstract
In Container Integration testing framework is a combination of four open source technologies. Arquillian, Test NG, JaCoCo, and Maven. The aim of this work is to deals with automating the code using JaCoCo. JaCoCo is a Java framework calculates code coverage. It find's the amount of code coverage in each lines of module that has been executed or missed and finally it will be deploy to wild fly server in the user matrix project source container.The main idea behind developing in this automation testing framework is able to test server side components developed using Java. The tests will be such that they will run in the container/application server (e.g. Wild Fly) where the server side component (e.g. test1) is deployed and because of that the tests will be able to use all the real resources (e.g. EJB etc.) provided by the container instead of mocking them.[1]Web Service Description Language (WSDL) specification, we first automatically generate necessary Java code to implement a client. We then leverage automated unit test generation tools for Java to generate unit tests, and execute the generated unit tests, which in turn invoke the service under test. The next important objective is to calculate amount of code covered by the test cases.

1. Introduction

In Container Integration testing framework is an integrated system that sets the rules of automation of a specific product which integrates several components such as function libraries, test data sources, object details and various reusable modules. These components help building a suitable automation framework which enable testing the business process as per the requirements. In addition, test automation is used to control the execution of tests and the comparison of actual outcomes with predicted outcomes[2]. Manual Testing is a type of Software Testing where Testers manually execute test cases without using any automation tools[3]. Automatic Testing technique where the tester writes scripts by own and uses suitable software to test the software. It is basically an automation process of a manual process.The procedure being utilized to execute automation is known as a test computerization system, a few structures have been actualized throughout the years by business sellers and testing association. Mechanization actualized when it has been resolved that the manual testing is not meeting desires, keep away from human mistakes and when it is impractical to get more manual analyzers.

Arquillian is a testing framework for Java that leverages JUnit and TestNG to execute test cases against a Java container.TestNG is a framework using which the test cases are written and these test cases are run by the Arquillian. TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration[6] JaCoCoframework calculates code coverage. The coverage report calculated by

JaCoCo not only gives ball park view of how much has or has not been covered by the test cases but also give a code level view, showing the covered code in green color, partially covered code in yellow color, and missed code in red color [7]. Maven is used to handle the dependencies required for running Arquillian, TestNG and JaCoCo. It has been used a tool to bind the rest of the technologies to work together.

2. Literature survey

Existing system has Manual Testing that uses unit Testing using mocks[4]. An object under test may have dependencies on other objects. To isolate the behavior of the object you want to test you replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test. This unit test is not isolated, it always depends on external resources like database. This unit test can't ensures the test condition is always the same, the data in the database may vary in time It's too much work to test a simple method, cause developers skipping the test.[5]Proposed System is In container integration testingArquillian helps simplify integration testing of application. It is designed by keeping to eliminate the drawbacks of the present system in order to provide the solution for the existing problems. The main focus is on.Reducing manual test work.Finding code coverage using JaCoCoIn-Container testing of JavaEEcomponents.In container Integration testing is aimed at automating the server side components and testing of all the java based products along with generation of code coverage report. Functional and non-functional web service testing [10] is done with the help of WSDL parsing and regression testing is performed by identifying the changes made thereafter. Web service regression testing needs can be categorized in three different ways, namely, changes in WSDL, changes in code, and selective re-testing of web serviceoperations.Representational State Transfer (RESTful) Web Services: The functionality for RESTful web services is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP based services are, do not require XML messages or WSDL service API definitions. It is noted that in all web service applications, the designed testing automation framework does not work efficiently as it requires human intervention and test data dependency [11].

***Corresponding Author,**
E-mail-address: sitvijay@gmail.com

3 Methodology

3.1 Design Description

This Automation testing framework 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. WildFly, so that the tests can also use the 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 also works as dependency management and build tool.

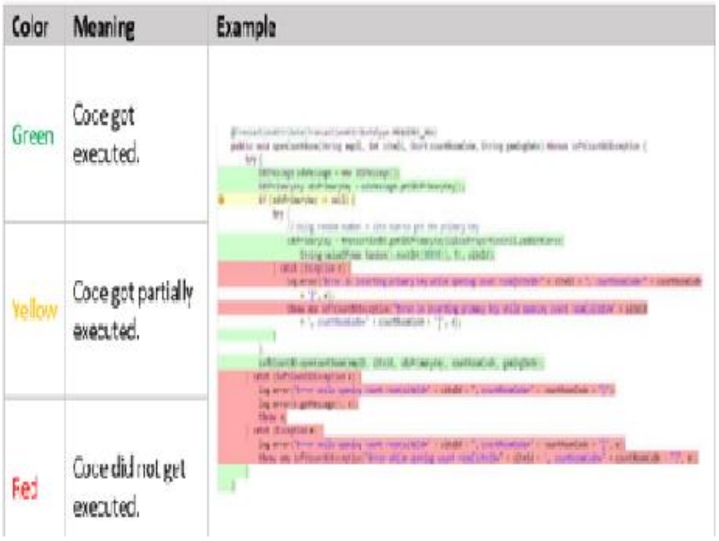
3.2 Structured Design

This is an industry standard. The technique starts by identifying inputs and desired outputs to create a graphical representation. Structured design has been adopted. 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. It shows executed methods and unexecuted methods

4 Design and Implementation

4.1 Design of the system

In container integration testingArquillian helps simplify integration testing of application. Unit tests live in their own world e.g. when you have to test your EJBs, you have to mock the EJB container and a whole lot of other things. With Arquillian you no longer have to do all those mocking or any other plumbing job to run your tests. Arquillian run your tests inside the choice of your container. It



Color	Meaning	Example
Green	Code got executed.	[illegible]
Yellow	Code got partially executed.	[illegible]
Red	Code did not get executed.	[illegible]

Fig: 1: Color coding of JaCOCO report

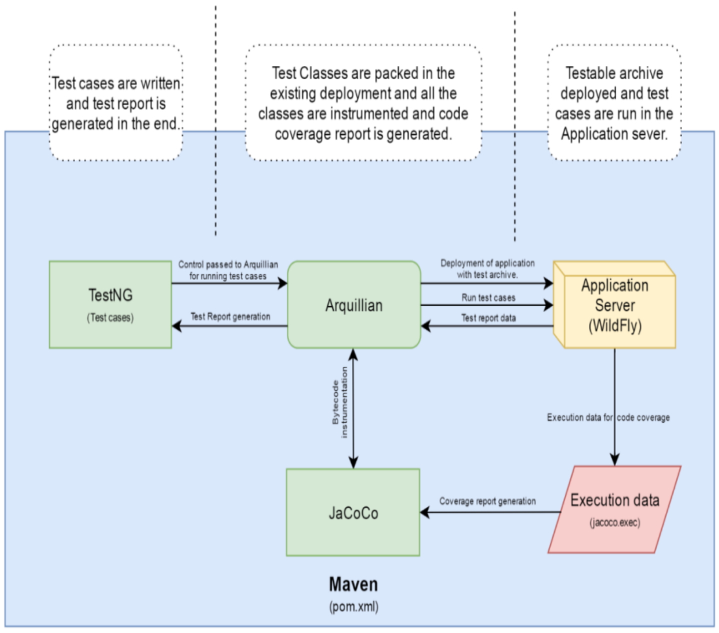

Test cases are written and test report is generated in the end.
Test Classes are packed in the existing deployment and all the classes are instrumented and code coverage report is generated.
Testable archive deployed and test cases are run in the Application sever.
TestNG
(Test cases)
Control passed to Arquillian for running test cases
Test Report generation
Arquillian
Deployment of application with test archive.
Run test cases
Test report data
Application Server (WildFly)
Bytecode instrumentation
Execution data for code coverage
JaCoCo
Coverage report generation
Execution data (jacoco.exec)
Maven
(pom.xml)

Fig. 2 : Architecture of In container integration frame

deploys your test code with your application code so that the test code can leverage all the services of the container and your application gets to live in the real world of its container.Include a diagram as follows and explain how it has been modeled to accomplish the user requirements. e.g. JaCoCo for code coverage, Arquillian for deploying and running test in the application server.build management maven has been used to configure test and application classes that need to be built, the container adapter (e.g. wildfly-arquillian-container-remote) to use when Arquillian will try to deploy the test code in an application container (e.g. WildFly), and JaCoCo to include/exclude application source files in the code coverage report.

Fig 4 DFD tells above the action performed by client Here, Fetching data from Data Base, and [9]creating a JSON file. Abstract Data Factory, were creating an Object for which is related to a method. Writing a test case using of data factory object and using values of data base to asserting. Test Case runs in a server. Final Step is to Check JaCoCo.

JaCoCo uses byte code instrumentation to modify the compiled classes in the archive (e.g. test1.ear) By modifying the compiled classes JaCoCo puts some markers at appropriate places in the byte code and when a line of byte code is executed that marker changes to indicate the same. This execution data is kept in a special file called jacoco.exec. After the execution of all the test cases, JaCoCo maps the execution data to the corresponding source files to generate the coverage report.

```
<build>
       <plugins>
              .
              .
              <plugin>
                     <groupId>org.jacoco</groupId>
                     <artifactId>jacoco-maven-plugin</artifactId>
                     <version>0.7.7.201606060606</version>
                     <executions>
                            <execution>
                                   <id>default-prepare-agent</id>
                                   <goals>
                                          <goal>prepare-agent</goal>
                                   </goals>
                            </execution>
                            <execution>
                                   <id>default-report</id>
                                   <goals>
                                          <goal>report</goal>
                                   </goals>
                            </execution>
                     </executions>
              </plugin>
              .
              .
       </plugins>
</build>
```

Fig 3: Adding Maven pom.xml dependency

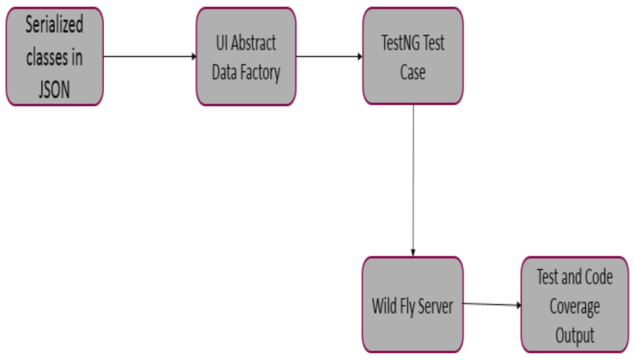
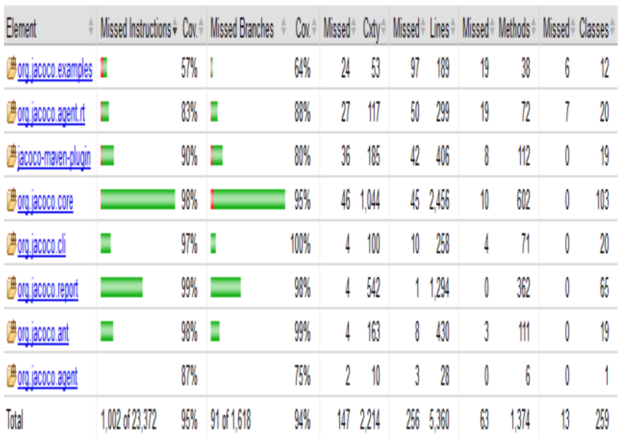


Fig. 4: Data Flow Diagram



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		57%		64%	24	53	97	189	19	38	6	12
org.jacoco.agent.rt		83%		88%	27	117	50	299	19	72	7	20
jacoco-maven-plugin		90%		80%	36	185	42	406	8	112	0	19
org.jacoco.core		98%		95%	46	1,044	45	2,456	10	602	0	103
org.jacoco.cli		97%		100%	4	100	10	258	4	71	0	20
org.jacoco.report		99%		98%	4	542	1	1,294	0	362	0	65
org.jacoco.ant		98%		99%	4	163	8	430	3	111	0	19
org.jacoco.agent		87%		75%	2	10	3	28	0	6	0	1
Total	1,002 of 23,372	95%	91 of 1,618	94%	147	2,214	256	5,360	63	1,374	13	259

Fig 5: JaCoCo Code Coverage Report

5. Results

5.1 Running test cases with code and cover report

It going to be a slow process so we are going to use it less often. This approach requires the application server running with no deployment in it. Arquillian will connect to the running application sever deploy the whole application along with the test classes and test cases will be run against the application. Covered test methods are represented by green color and uncovered by red color

6. Conclusions

Aimed at automating server side testing of all the java based products along with generation of code coverage report. The technology industry is moving towards automating everything it can. The major chunk of that automation includes automating the manual testing so that the products can be tested thoroughly, comprehensively, and as fast as possible. Server side testing can be automated. This will save time of QA so that they can work hard to break the system by doing some rigorous testing and hence improving the quality of software. The coverage report generated by JaCoCo the developer can easily find the junk piece of code and remove it, keeping the code base clean. If some part of the application code is not getting covered then one can write test cases to cover that part of code making the additions of code more safe and reliable.

7. References

[1]. X Bai, W Dong, WT Tsai, Y Chen. WSDL-based automatic test case generation for web services testing. In Proc. IEEE International Workshop on Service-Oriented System Engineering, 2005, 215–220.
[2].http://www.guru99.com/manual-testing.html
[3].http://www.softwaretestingclass.com/what-is-automation-testing/
[4].https://www.mkyong.com/unittest/unit-test-what-is-mocking-and-why/
[5]. Book: Arquillian Testing Guide by John D. Ament
[6].TestNG official documentation: http://testng.org/doc/documentation-main.html
[7]Maven official documentation:
https://maven.apache.org/guides/
[8] http://www.jacoco.org/jacoco/trunk/coverage/
[9] ttps://www.tutorialspoint.com/json/json_objects.htm
[10] http://en.wikipedia.org/wiki/Web_service
[11] MI Ladan. Web Services Testing Approaches: A Survey and a Classification, 2010, 70–79