

International Institute of Information Technology, Bangalore



Project Elective / Reading Elective Report

Project Title: Testing Automation tools

Mentor/Guide

Prof. B. Thangaraju,

Gurleen Kaur
Reg.No: IMT2017020

Rohan Mehta
Reg.No: IMT2017035

Abstract

The project aims at analyzing various open-source testing automation tools and comparing them in terms of their available documentation, ease of use, maintainability, reliability, extensibility, language support, integration with CI servers like Jenkins, functionality and various other factors.

Introduction

Testing is an integral part of the Software Development Life Cycle. Any software needs to be tested before it is deployed. Automation testing is a common thing for large enterprises, but most of the small and medium-sized companies resort to only doing manual testing. This is because there is a common misconception that automation testing is laborious, time-consuming and does not have a good return on investment. In the longer run, in fact, manual testing becomes extremely difficult and tends not to detect bugs that can be easily identified using automation testing. Over the past few years, there has been a surge in open source automation testing tools which require minimal effort and are highly extensible. With over 100 proprietary and open source automation testing tools available, it becomes tough to choose the right automation tool for your software. One needs to understand the project requirements thoroughly before choosing a testing automation tool. Choosing the right testing automation tool is a crucial decision, and if taken correctly, can help save a lot of time and give an excellent return on investment.

The project was undertaken to explore the various open-source automation testing tools and compare them in terms of their available documentation, ease of use, maintainability, reliability, extensibility, language support, integration with CI servers like Jenkins, functionality and various other factors. It aims to help people choose the right testing automation tool that fulfils their software's requirements.

We completed the project in 3 broad phases :

Phase 1

In phase 1, we built a Maven Swing Calculator Application. We built the app with UI so that it could be used to test various functional UI testing tools as well.

Phase 2

We shortlisted some of the testing tools that supported our machine's hardware and software requirements, as well as our Calculator software's requirements. We chose a variety of testing tools ranging from unit testing to functional testing to load and stress testing. We chose some popular tools and some not so popular but useful tools. We tried these tools on our desktop calculator application built in phase 1. Some of the testing tools, like Selenium, that were compatible only with web applications were tested on our college's moodle URL - lms.iiitb.ac.in. After trying out these tools, we compared them using various factors.

Phase 3

Since the usage of continuous integration and deployment has become a common practice, we wanted to check these testing tools compatibility with one of such CI/CD servers, Jenkins. We created a Jenkins pipeline. The source code of the java Calculator Application was on Github. We created webhooks to link the Github repository to the Jenkins server. The pipeline works as follows - After pushing the code on the Github repository, the pipeline is initiated. Then, the pushed code is pulled by the Jenkins server, and a jar is built using the maven plugin. Junit is used to perform the unit testing and tools like Jubula, SikuliX are used for automated functional GUI testing. Post the success of these tests; a report is generated and published on the Jenkins server. We were using the local machine for running the tests, and the UI testing tools required the system's display and mouse clicks. This interfered with our working on the local system. Therefore, we used the xvfb plugin for Jenkins, which is an in-memory display server for Linux. It implements the X11 display server protocol and runs all the GUI operations in virtual memory without any output on the screen while also having the ability to take screenshots [1].

System Configuration

The specifications of the system used for this work are :-

Operating system	Linux Ubuntu
Version	16.4
Kernel version	4.4.0.116 -generic
CPU cores	8
RAM	16 GB

Installation Procedure

The project required installation of Java and Eclipse IDE for Java Developers to create a Maven Calculator Application. Later in the second phase it required installation of various testing tools namely Jubula, Selenium, SikuliX and JMeter. Installation of Jenkins and various Jenkins plugins was required in the third phase.

Phase 1

Installing Java and Eclipse IDE for making the calculator Desktop Application

Installing Java

- Check if java is already installed, by running

```
$ java -version
```

- If not installed, run

```
$ sudo apt install openjdk-8-jdk
```

Installing Eclipse IDE for Java developers

- Download the latest version of eclipse IDE for Java developers from this link.

<https://www.eclipse.org/downloads/packages/>

- Once downloaded extract the tar folder, open the source folder and run the executable by double clicking on it.
- Eclipse IDE launcher appears on the screen, then select a directory as workspace and click on launch (refer Figure 1.1). The eclipse IDE will open.

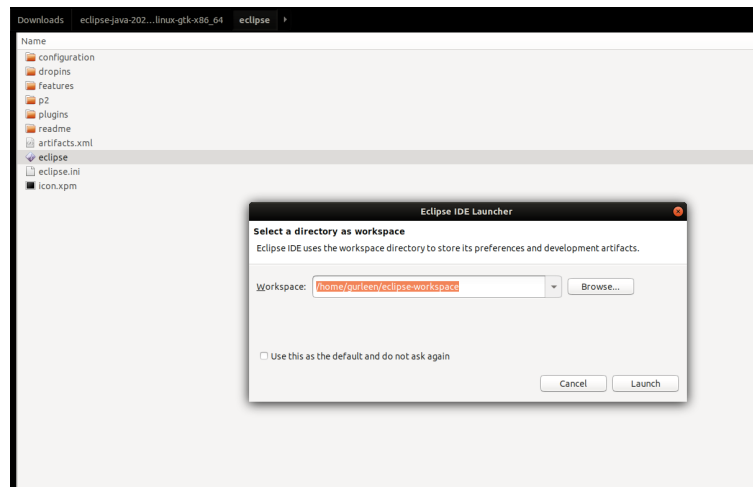


Figure 1.1 Eclipse Launcher

Phase 2

Installing various testing tools.

Setting up Junit5 in a maven project

- Open the maven Calculator project in eclipse and go to pom.xml file.
- In the pom.xml file, under the dependencies section, add the following lines of code (ref Code 1) [3]

```

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.5.0</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.platform</groupId>
  <artifactId>junit-platform-runner</artifactId>
  <version>1.5.2</version>
  <scope>test</scope>
</dependency>

```

Code 1: Maven Junit5 dependency code

- After adding this to pom.xml, go to the project directory and open terminal and run

```
$ mvn package
```

- If maven is installed, it will package successfully, else will give an error saying - The program 'mvn' is currently not installed.
- To install maven, open terminal and run

```
$ sudo apt install maven
```

Installing Jubula

- Download Jubula Standalone by going to this link
<https://testing.bredex.de/jubula-download-page.html>

- Once downloaded go to command line and run

```
$ cd Downloads && bash installer-jubula_linux-gtk-x86_64.sh
```

- After running this Jubula wizard opens. Follow the wizard, accept the agreement and set the workspace directory. The wizard gives an option to select which Jubula components to install (refer Figure 1.2).

We require Jubula and AUT Agent. But it is good to have the documentation as well. Otherwise, the documentation can also be accessed online. Keep following the wizard and Jubula will be set up.

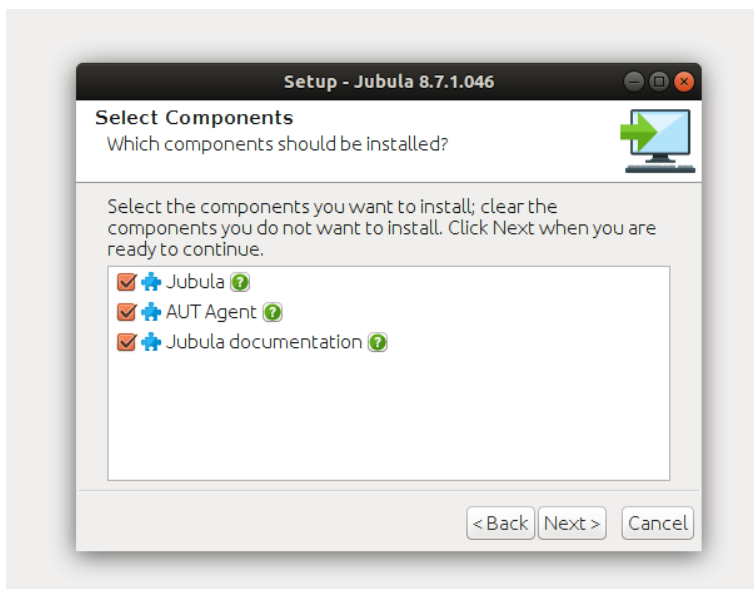


Figure 1.2 Jubula Setup Wizard

Installing SikuliX

- Download sikulix.jar from this link.
<https://raiman.github.io/SikuliX1/downloads.html>
- Just download and move it to a folder in Desktop.
- Once moved, go to the folder and then open command line and run

```
$ java -jar sikulix.jar
```
- You can simply double click on the sikulix.jar folder to run it. If the installation is successful, SikuliX IDE will launch (ref Figure 1.3)

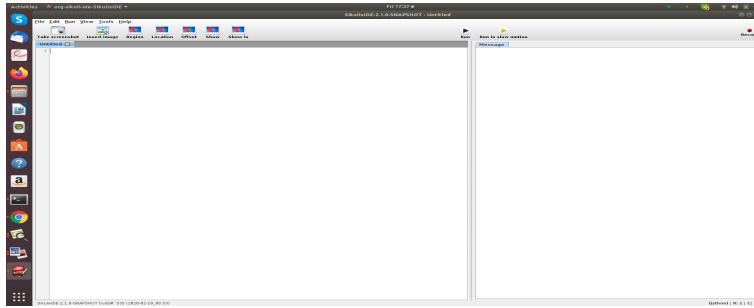


Figure 1.3 SikuliX IDE

Installing Selenium

We would add the Selenium dependency to a maven project. To do this ->

- Create a simple Java Maven project in Eclipse. (refer Experimental Setup - Phase 1)
- Now open the pom.xml file, under the dependencies section, add the following lines of code (Refer Code 2)

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>
```

Code 2: Maven Selenium dependency code

- After adding this to pom.xml, go to the project directory and open terminal and run

```
$ mvn package
```

- To check if Selenium is correctly installed, restart the Eclipse IDE and open your project. Under the project directory, click on the Maven Dependencies directory. If Selenium was set up successfully, the dependencies would include Selenium JAR files.

Installing JMeter

- Download JMeter by going to this link.
<https://jmeter.apache.org/>
- Once downloaded, extract the package folder, move this extracted folder to your Desktop.
- Once extracted open the terminal at the same location and run

```
$ cd apache-jmeter-5.2.1/bin/ && bash jmeter.sh
```
- If JMeter is installed properly. On running the above command would open JMeter GUI application. (refer Figure 1.4)

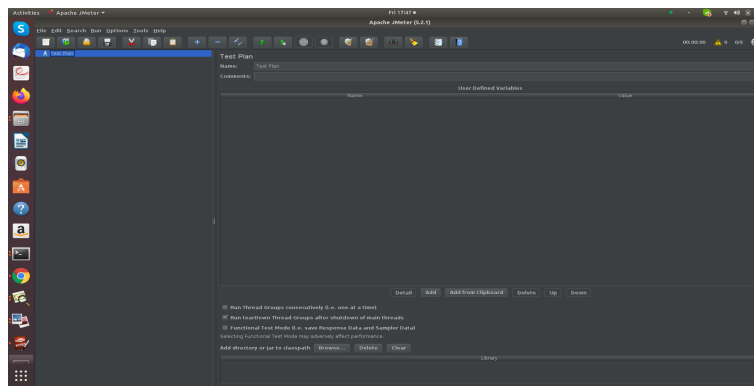


Figure 1.4 Apache JMeter

Phase 3

Installing Jenkins [22]

- Firstly add the repository key to the system, Use the command line and RUN

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -
```
- After the key is added the system will return OK.
- Then add the debian package repository address to the server's sources.list

- Then run update so that apt-get will use the new repository.
- Now use the command line and RUN
sudo apt-get install jenkins
- Now use the systemctl command to start the Jenkins.
sudo systemctl status jenkins
- Finally to set up the installation, visit Jenkins on its default port, 8080, using the server domain name or IP address:
http://ip address or the domain name :8080

Installing various Jenkins plugins

To install Jenkins plugins ->

- Go to Jenkins dashboard.
- On the left side of the screen, click on Manage Jenkins button.
- A new screen opens. Scroll down and click on Manage Plugins.
- After clicking on Manage Plugins, a screen with 4 tabs, namely Updates, Available, Installed and Advanced opens. (Refer Figure 1.5). Select the Available tab. A list of plugins appears (Refer Figure 1.5)

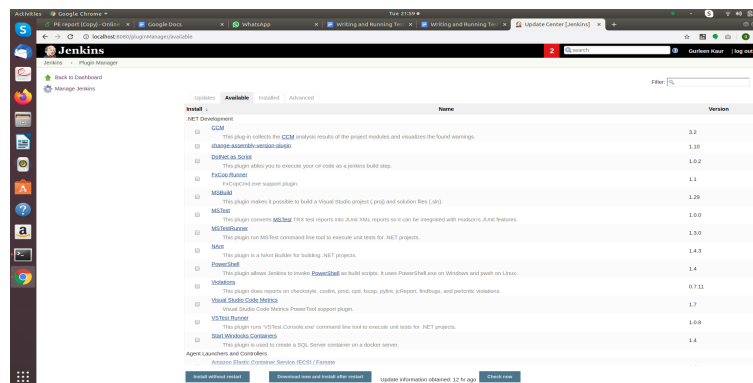


Figure 1.5 Jenkins Manage Plugins Screen

- Click on the plugins required - we required Maven integration, Junit, Xvfb, HTMLPublisher, and GitHub. After selecting them, click on Install without restart. (Refer figure 1.5)

Experimental Setup

The experimental setup would also be discussed in 3 phases

Phase 1

Creating the Java Maven Calculator Application

- Open the eclipse IDE. To do this, go to the folder where you had downloaded eclipse and run `./eclipse` on command line as seen in the Figure 2.1

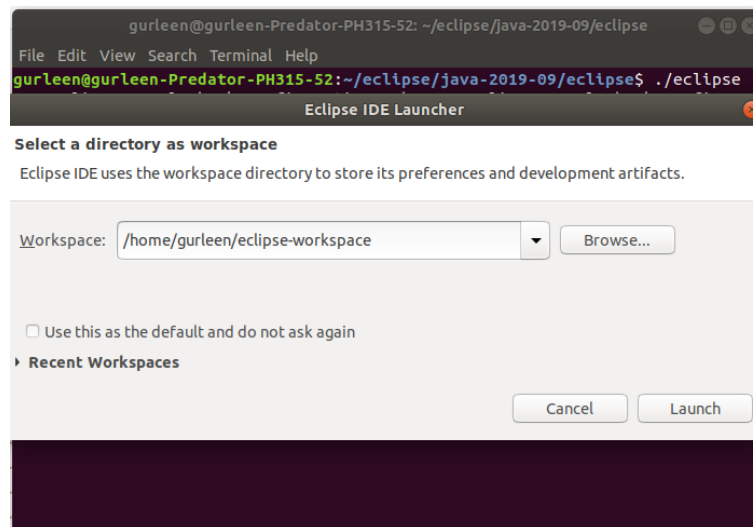


Figure 2.1 Running eclipse from command line

- Eclipse IDE launcher appears as shown in Figure 2.1. Click on the launch button to launch the IDE
- Once the Eclipse IDE opens, click on File on menu bar, then click on new and then on other.
- A modal window appears (refer Figure 2.2). Select Maven Project under the Maven folder and click on Next. Keep following the New Maven Project wizard. Enter group id and artifact id of the project and click on finish.

- The basic maven project boilerplate code gets created with a standard pom.xml file.
- Next, We create the GUI Calculator application using Swing API.

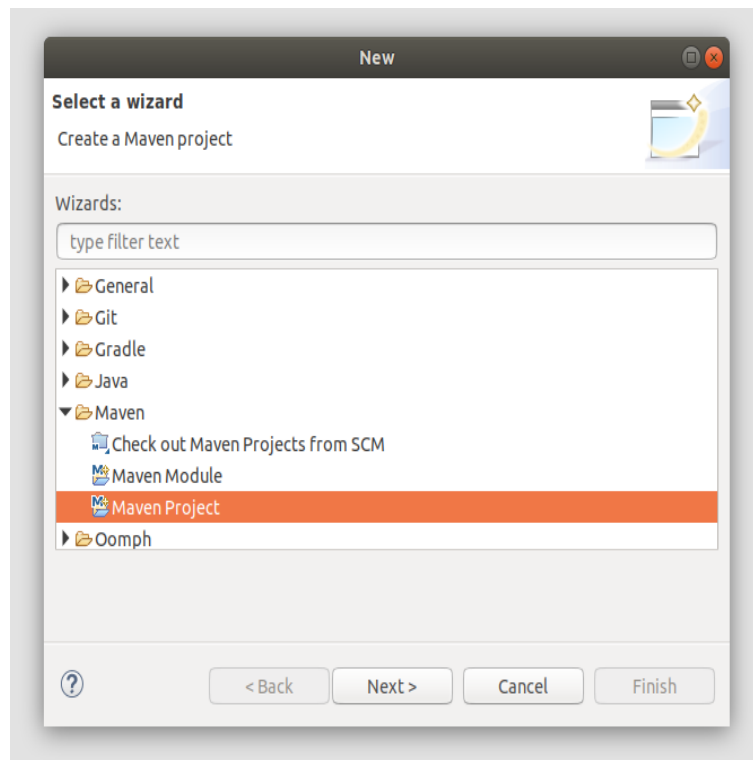


Figure 2.2 Configuring Maven Project

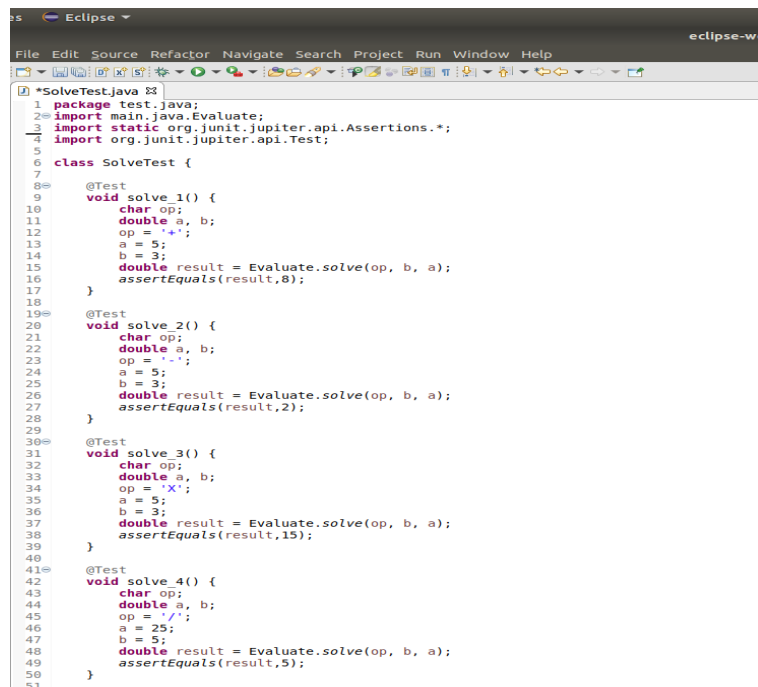
Phase 2

Writing and running Junit5 tests

- Open the source project in Eclipse. Go to src/test/java and create a new file, say SolveTest.java.
- Import the class that has functions you want to unit test. Also add the following imports to the top of your file.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
```

- Create the test class, and start writing test cases in the form of test methods. To define that a method is a Test method annotate it with @Test Annotation. JUnit provides static methods to test for certain conditions via the Assert class. [4]
- As you can see in Figure 2.3, assertEquals is used to check if the actual result is equal to the expected result. Other assertions like assertTrue, assertNull etc can also be used. Functioning of various operators was tested for the calculator app.



```

1 package test.java;
2 import main.java.Evaluate;
3 import static org.junit.jupiter.api.Assertions.*;
4 import org.junit.jupiter.api.Test;
5
6 class SolveTest {
7
8     @Test
9     void solve_1() {
10         char op;
11         double a, b;
12         op = '+';
13         a = 5;
14         b = 3;
15         double result = Evaluate.solve(op, b, a);
16         assertEquals(result, 8);
17     }
18
19     @Test
20     void solve_2() {
21         char op;
22         double a, b;
23         op = '-';
24         a = 5;
25         b = 3;
26         double result = Evaluate.solve(op, b, a);
27         assertEquals(result, 2);
28     }
29
30     @Test
31     void solve_3() {
32         char op;
33         double a, b;
34         op = 'x';
35         a = 5;
36         b = 3;
37         double result = Evaluate.solve(op, b, a);
38         assertEquals(result, 15);
39     }
40
41     @Test
42     void solve_4() {
43         char op;
44         double a, b;
45         op = '/';
46         a = 25;
47         b = 5;
48         double result = Evaluate.solve(op, b, a);
49         assertEquals(result, 5);
50     }
51 }

```

Figure 2.3 JUnit Test Class

- To run the Junit5 test cases, open your terminal at the source project and run

```
$ mvn test
```

- As you can see in Figure 2.4, the output shows the total test cases run and failures or errors if any.

```
gurlleen@gurlleen-Predator-PI315-S2:~/Desktop/testing_automation/Testing_automation/Java-Calculator/calculator$ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ----- calculator:calculator -----
[INFO] building calculator 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ calculator ---
[INFO] using 'UTF-8' encoding to copy filtered resources.
[INFO] copying 2 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ calculator ---
[INFO] using 'UTF-8' encoding to copy filtered resources.
[INFO] copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ calculator ---
[INFO] Changes detected - recompiling the module!
[INFO] compiling 2 source files to /home/gurlleen/Desktop/testing_automation/Testing_automation/Java-Calculator/calculator/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ calculator ---
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running test.java.SolveTest
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.024 s - in test.java.SolveTest
[INFO] Running test.java.EvaluateTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in test.java.EvaluateTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.069 s
[INFO] Finished at: 2020-05-10T02:31:35+05:30
[INFO]
[INFO] -----
```

Figure 2.4 Junit5 test output

Writing and running Jubula tests

Writing Jubula tests involves 3 main steps.

- **Configuring the Application Under Test (AUT)**

- Open the Jubula ITE (refer Figure 2.8). On the menu bar click on Test and then on New. New Project Wizard Opens.
- Write the project name say, Calculator, From the drop-down, select project toolkit as Swing. This is because we have built a Swing Java Calculator Application. Next, Click on Finish.
- Now press Ctrl + Shift + P on the keyboard. A screen titled Properties for Calculator opens. (refer Figure 2.5)
- Under AUTs defined for this project section, double click on the AUT created.(refer Figure 2.5)
- AUT (Application Under Test) modal opens. To configure the AUT, double click on Calculator@localhost under the AUT configurations.
- AUT configuration modal opens (refer Figure 2.6). Click on the Expert button on this screen. To enter the Executable JAR File Name, browse for the Java Calculator's JAR file. This can be

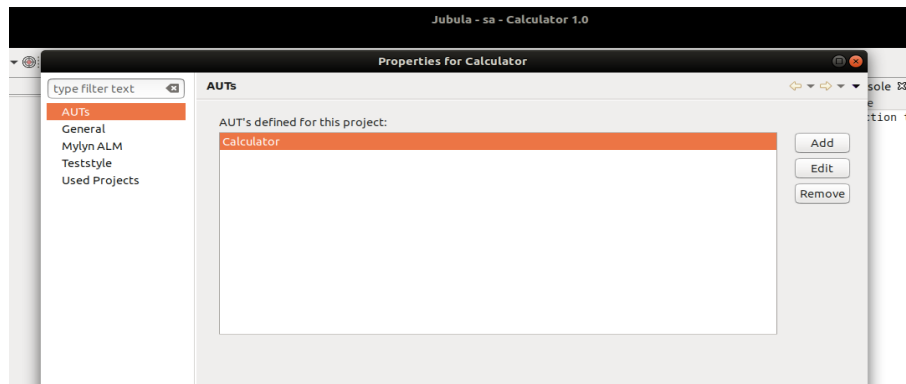


Figure 2.5 Properties for Calculator modal

found in the Java Calculator project's 'target' folder. Next to enter the AUT working directory, enter the path for the 'target' folder present inside the Java Calculator Project. Finally in the AUT arguments section type **-jar <JAR file name>**

- After doing this click on Try AUT. (refer Figure 2.6)

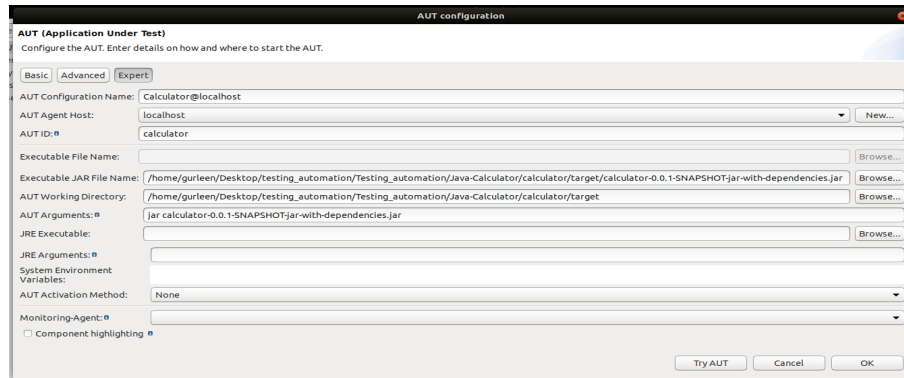


Figure 2.6 AUT Configuration

- If the configuration is correct, the calculator shows up on the screen.

• Object Mapping

Object mapping is the process by which we join the names of the components that we would like to use in our test cases to the real components of the Application Under Test (AUT).

- To do object mapping we need to first start our AUT.
- We will first connect to the AUT agent. Click on the connect to AUT agent button present on the toolbar (refer Figure 2.7 - 4th button) and then click on connect to embedded AUT agent.
- Now press Ctrl + R to run the AUT
- Click on start object mapping mode button present on the toolbar (refer Figure 2.7 - circular button with green center and black border).

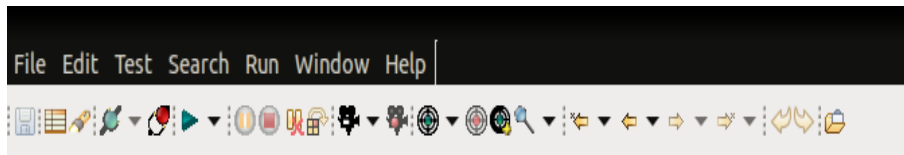


Figure 2.7 Jubula MenuBar and ToolBar

- This opens an Object Mapping View (refer Figure 2.8). This has 2 Columns Unassigned component names and Unassigned technical names.
- Suppose you want to refer the button displaying number 6 as button6 in the test cases. To do this open the AUT, hover over the desired button and press Ctrl + shift + Q. This will add the component to the Unassigned technical names column.
- Now tap on the unassigned component names column, and select New Component Name option. Enter the desired name, in this case button6. This creates an entry for button6 in the Unassigned component names column.
- Drag and drop this to the entry created in the Unassigned technical names column.
- Now the button can be accessed by the name button6 in the test cases. Similarly the other components can be mapped.

• Writing Test Cases

- Go to the Test Case Browser (refer Figure 2.8). Right click , go to new can click on New Test Case.

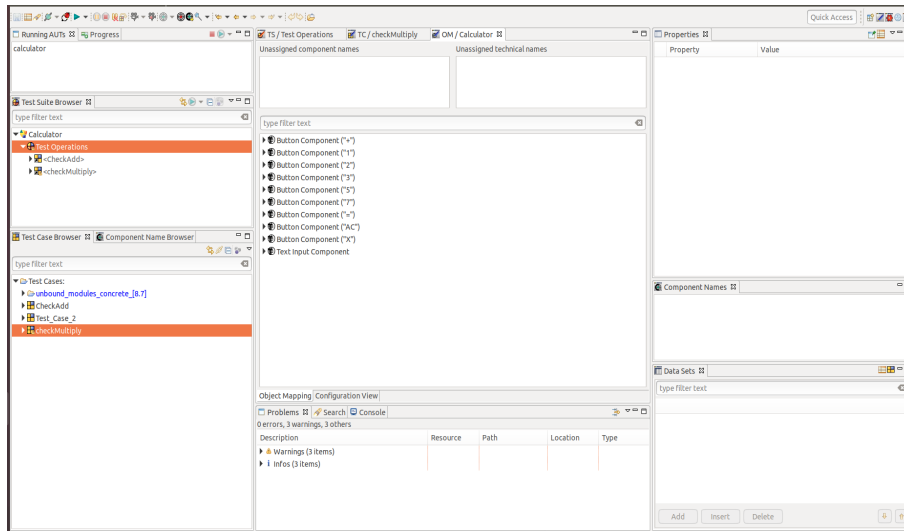


Figure 2.8 Jubula ITE screen

- Enter the name of the test case and click on okay.
- Now double click on the test case name. The Test case window appears
- Right click and then click on New and click on ‘New Test Step’.
- The New Test Step modal appears (refer Figure 2.9). Add the Test Step name, select component type. In the Component name write the name used while creating the object mapping. Select an action (click, check, hover etc) and click on Finish.

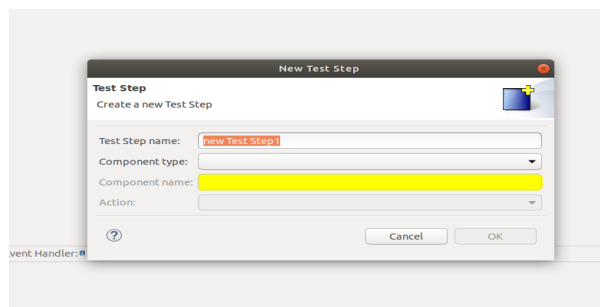


Figure 2.9 New Test Step Modal

- Similarly add more test steps. Once the test case is completed drag it to the Test Suite Browser Window.
- Similarly, other test cases can be added as well.
- Your Test Suite is now ready to be run. To run the test suite, go to the Test Browser section of the screen and click on the green play button present there. (Refer Figure 2.8). The test cases can also be run from command line using the testexec command.
- The report of the test suite being run can be seen on the right side of the screen. If the test suite is run from the command line, reports get saved to the specified result directory.

Writing and running SikuliX tests

- Run the sikulix.jar file to launch the IDE.
- Click on File and then New.
- Now launch the Java Calculator Application
- Start writing the script. Use the Take Screenshot button (refer Figure 2.10) to let the script recognize the component it wants to interact with.

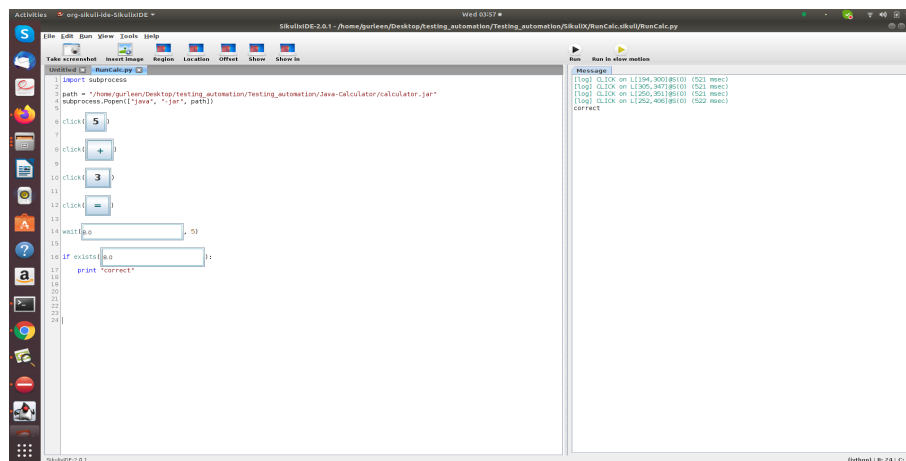


Figure 2.10 SikuliX IDE and Script

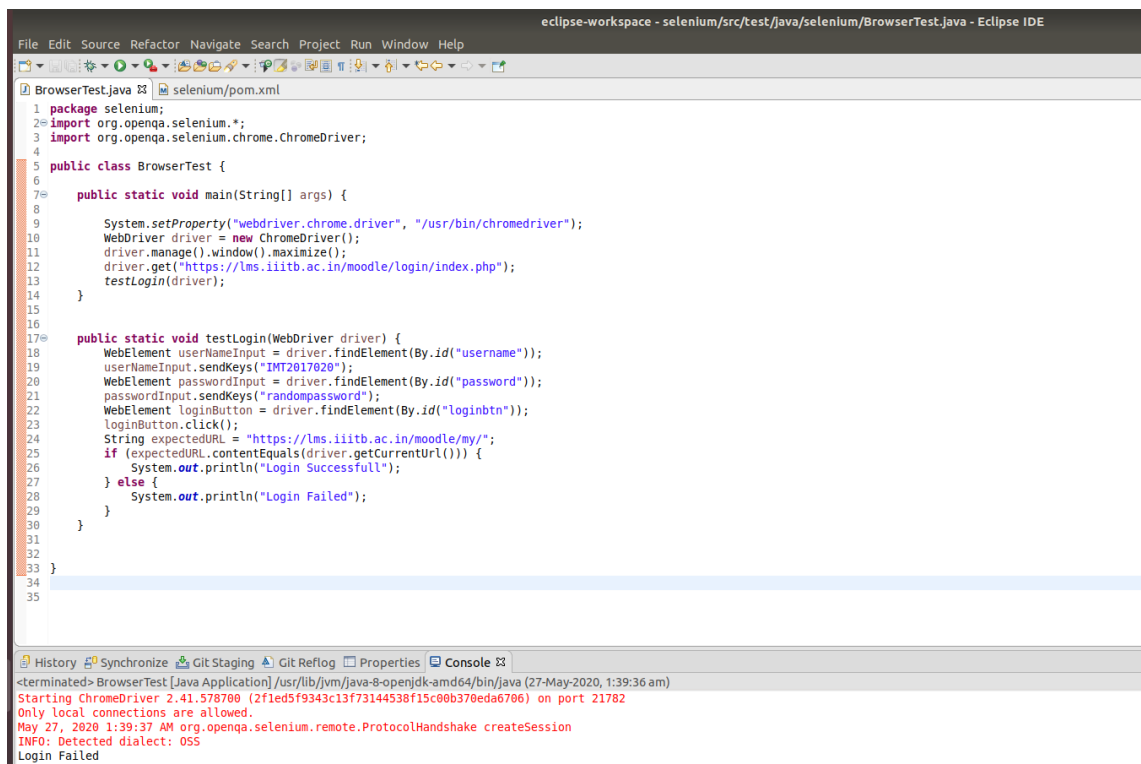
- The Application Under Test needs to be launched via the script before writing the tests. As you can see in Figure 2.10, the calculator jar file is launched.
- After the Jar file is launched, button clicks take place. It waits for some time for the result to appear. If the result exists then it prints correct. (refer Figure 2.10)
- To run the script press Ctrl + R.

Writing and running Selenium tests

- Open the Selenium project (refer Installation - Phase 2 - installing Selenium) in eclipse.
- Next, go to src/test/java and create a file, say BrowserTest.Java
- Add the following imports to the top of this file.

```
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
```

- Main function is where all the test cases will be run. Inside the main function, help the webdriver find the downloaded ChromeDriver executable by specifying its location via the 'webdriver.chrome.driver' system property [13] (Refer Figure 2.11 - line 9).
- Next, create an instance of the WebDriver interface and cast it to ChromeDriver class (Refer Figure 2.11 - line 10).
- Maximise the Chrome Browser screen. (Refer Figure 2.11 - line 11)
- Pass the URL at which the website you want to test is present in the driver.get function. (Refer Figure 2.11 - line 12)
- To write the first test case, create a function for testing login functionality of IIITB LMS.
- Use the ID locator to locate the Username input, Password input and the login button. (Refer Figure 2.11 - lines 18,20,22)

The screenshot shows the Eclipse IDE interface. The top part is the 'Editor' window displaying the 'BrowserTest.java' file. The code is as follows:

```
1 package selenium;
2 import org.openqa.selenium.*;
3 import org.openqa.selenium.chrome.ChromeDriver;
4
5 public class BrowserTest {
6
7     public static void main(String[] args) {
8
9         System.setProperty("webdriver.chrome.driver", "/usr/bin/chromedriver");
10        WebDriver driver = new ChromeDriver();
11        driver.manage().window().maximize();
12        driver.get("https://lms.iiitb.ac.in/moodle/login/index.php");
13        testLogin(driver);
14    }
15
16
17    public static void testLogin(WebDriver driver) {
18        WebElement userNameInput = driver.findElement(By.id("username"));
19        userNameInput.sendKeys("IMT2017020");
20        WebElement passwordInput = driver.findElement(By.id("password"));
21        passwordInput.sendKeys("randompassword");
22        WebElement loginButton = driver.findElement(By.id("loginbtn"));
23        loginButton.click();
24        String expectedURL = "https://lms.iiitb.ac.in/moodle/my/";
25        if (expectedURL.equals(driver.getCurrentUrl())) {
26            System.out.println("Login Successful");
27        } else {
28            System.out.println("Login Failed");
29        }
30    }
31
32
33 }
34
35
```

The bottom part of the IDE is the 'Console' tab, which shows the following output:

```
<terminated> BrowserTest [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (27-May-2020, 1:39:36 am)
Starting ChromeDriver 2.41.578790 (2f1ed5f9343c13f73144538f15c00b370eda6706) on port 21782
Only local connections are allowed.
May 27, 2020 1:39:37 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: OSS
Login Failed
```

Figure 2.11 Selenium Example Test Case

- Once the web elements are found, use 'sendKeys' function to send the values for email and password. (Refer Figure 2.11 - lines 19,21)
- Next, use 'click' function to click on the identified login button. (Refer Figure 2.11 - line 23)
- To check if the login was successful, compare the expected URL to actual URL of the web page upon clicking Login button. If the expected and actual URL are equal, login is successful, else login failed. (Refer Figure 2.11 - lines 24-28)
- Finally call this 'testLogin' function (our first test case) in the main function. (Refer Figure 2.11 - line 13)
- Save the file and run it. Once it finishes running, check the logs in the Console tab of the pane present below to know if the test case passed or failed. (Refer Figure 2.11)

Writing and running JMeter tests

- Firstly change the name of the Test Plan to LMS to do the Load Testing using Apache JMeter on the LMS website.
- Now to add thread groups, right click on the 'LMS' button present on the left panel and select add threads option.(Refer Figure 2.12) Thread groups represents the number of users.

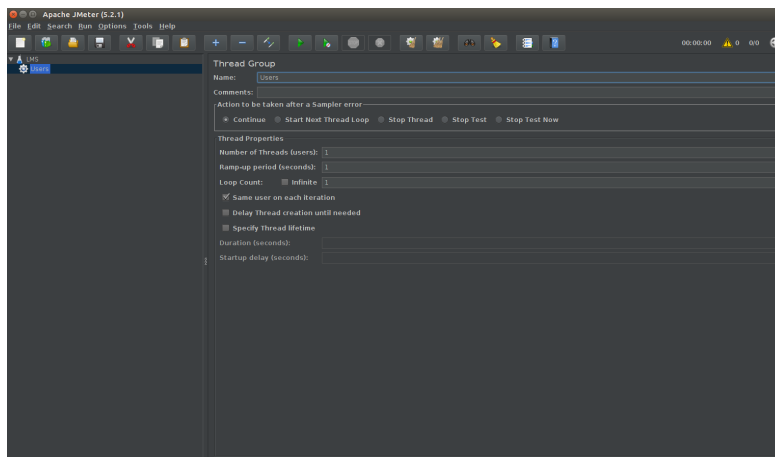


Figure 2.12 Adding Thread groups - Apache JMeter

- As Apache JMeter supports HTTPs requests, add samplers which is basically required for sending HTTPs requests. To add samplers, right click on the 'Users' button present on the left panel, under the 'LMS' button, and select add samplers option.
- Then give the server name and the path where we are going to send the HTTPs requests. (Refer Figure 2.13)
- Finally add listeners which are required to view the results in tabular or graphical format and then RUN the script by clicking on the green play button.
- Results can be seen in tabular format. (Refer Figure 2.14)

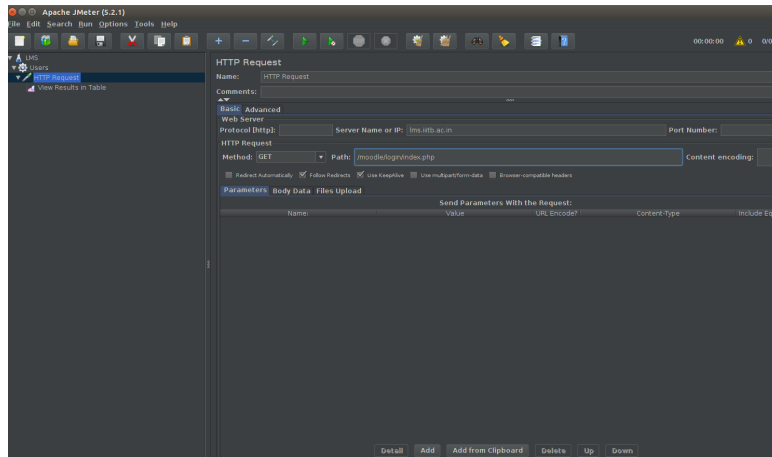


Figure 2.13 HTTP Request Screen - Apache JMeter

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	22:16:15.205	Users 1-1	HTTP Request	1007	✓	28492	284	405	322
2	22:16:26.934	Users 1-1	HTTP Request	457	✓	28492	284	110	52
3	22:16:30.016	Users 1-2	HTTP Request	926	✓	28492	284	109	53
4	22:16:30.228	Users 1-4	HTTP Request	1446	✓	28492	284	104	51
5	22:16:30.309	Users 1-5	HTTP Request	2081	✓	28492	284	105	52
6	22:16:30.513	Users 1-7	HTTP Request	2033	✓	28492	284	132	76
7	22:16:30.714	Users 1-9	HTTP Request	2950	✓	28492	284	1273	69
8	22:16:30.808	Users 1-10	HTTP Request	3084	✓	28492	284	1026	60
9	22:16:30.123	Users 1-5	HTTP Request	3956	✓	28492	284	105	52
10	22:16:30.411	Users 1-6	HTTP Request	3645	✓	28492	284	116	62
11	22:16:30.612	Users 1-8	HTTP Request	4141	✓	28492	284	385	76

Figure 2.14 Results - Apache JMeter

Phase 3

Creating the Jenkins Pipeline.

- Open Jenkins on browser and login.
- Next, click on the New Item button, present to the left side of the screen.
- Enter the Name and select the 'Pipeline' option and click on OK.
- Next, click on the GitHub option and enter the GitHub URL of the project.
- Under Build Triggers click on GitHub hook trigger for GITScm polling.
- Under the pipeline option, add the repository url and git credentials. Also under script path write 'JenkinsFile'. We would be writing the pipeline script in this file. Click on save.
- **Creating the JenkinsFile ->**
- The Jenkins Pipeline will have 3 stages namely Checkout SCM, Maven Build and Test.
- Add maven inside the tools block. (refer Figure 2.15 - line 4)
- Next start a stages block, Inside the stages block, add three stage blocks. The 'checkout scm' will checkout code from source control; scm is a special variable which instructs the checkout step to clone the specific revision which triggered this Pipeline run [14]. (Refer Figure 2.15 - lines 11 - 15)
- After checking out to updated code, stage 'Maven Build' starts. In this stage, the calculator application is installed using the 'mvn clean install' command. (Refer Figure 2.15 - line 20)
- Post the success of the build, 'mvn site' command is run. This command runs the Junit test cases and generates a HTML report. Next using publishHTML, the reports are published to the Jenkins server (Refer Figure 2.15 - line 22-34)

```

Jenkinsfile x
Jenkinsfile
1 pipeline {
2   agent any
3   tools {
4     maven 'maven'
5   }
6   options {
7     skipDefaultCheckout(true)
8   }
9
10  stages {
11    stage('Checkout SCM') {
12      steps {
13        echo '> Checking out the source control ...'
14        checkout scm
15      }
16    }
17    stage('maven build') {
18      steps {
19        echo '> Building the maven application ...'
20        sh 'cd Java-Calculator && cd calculator && mvn clean install'
21      }
22      post {
23        success {
24          sh 'cd Java-Calculator && cd calculator && mvn site'
25          echo "build successful"
26          publishHTML target: [
27            allowMissing: false,
28            alwaysLinkToLastBuild: false,
29            reportDir: 'Java-Calculator/calculator/target/site',
30            reportFiles: 'surefire-report.html',
31            reportName: 'JUnit Test Report'
32          ]
33        }
34      }
35    }
36
37    stage('Jubula tests') {
38      steps {
39        wrap([{$class: 'Xvfb', autoDisplayName: true, additionalOptions: '', assignedLabels: '', displayNameOffset: 0, installa
40          echo "Running Jubula tests"
41          sh 'cd /home/gurleen/jubula_8.7.1.046/ite && chmod 777 * && ./testexec -project "Calculator" -version "1.0" -tes
42          echo "test successful"
43        }
44
45        publishHTML target: [
46          allowMissing: false,
47          alwaysLinkToLastBuild: false,
48          reportDir: 'Jubula',
49          reportFiles: 'reports.html',
50          reportName: 'Jubula UI testing report'
51        ]
52      }
53    }
54  }
55 }
56
57

```

Figure 2.15 JenkinsFile

- Next stage is for GUI testing, in this project's case Jubula testing. Xvfb wrap is used to run the tests in an in-memory display server for Linux. It implements the X11 display server protocol and runs all the GUI operations in virtual memory [1]. Inside the wrap `./testexec` command is used to run the Jubula tests. The project name, version, testsuite, datadir, dburl, dbuser, resultdir, and resultname arguments are passed as arguments to the testexec command. (Refer Figure 2.15 lines - 38-43)
- Again, using `publishHTML`, the Jubula test reports are also published to the Jenkins server. (Refer Figure 2.15 - line 45-51)

Results and Discussion

We tried various testing tools. The results and findings from each tool are presented below.

Junit5

Junit is one of the most popular testing tool for Java Frameworks. Junit5 is the latest version of Junit, released to provide support for all additional features of Java 8. JUnit 5 is composed of several different modules from three different sub-projects. JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage [5]

- It has a very comprehensive and easy to follow documentation.
- It can only be used to test Java Frameworks
- Test cases can also be written only in Java.
- It can be used for unit testing as well as integration testing.
- It is extensible. Uses the Extension API. Extensions can be registered declaratively via `@ExtendWith`, programmatically via `@RegisterExtension`, or automatically via Java's ServiceLoader mechanism.[6]
- It is very easy to use if one is comfortable with Java.
- It is reliable to the extent that code coverage can be measured using Eclipse. By running code coverage as Junit Test, the amount of code tested by a test class can be analysed. This ensures that maximum cases are considered and number of bugs can be minimised. Also, JaCoCo a free code coverage library for Java, created by the EclEmma team helps in generating code coverage reports[7]. Refer Figure 3.1
- Many companies including Apple, Intuit and Cisco use Junit.[8] The Junit Github repository is well maintained and the community is very active too. Junit5 is currently being used by 65.6K users and has 136 contributors [9]
- HTML reports can be generated using the maven-surefire plugin.

Element	Coverage	Covered Instructi	Missed Instructions
▼ Evaluate.java	11.6 %	29	222
▼ Evaluate	11.6 %	29	222
evaluate(String)	0.0 %	0	203
hasPrecedence(char, char)	0.0 %	0	16
solve(char, double, double)	100.0 %	29	0
▼ test.java	0.0 %	0	217
EvaluateTest.java	0.0 %	0	115

Figure 3.1 Code Coverage

Jubula

Jubula is an automated GUI functional testing tool. It is available as a part of the Eclipse package and also as a standalone. We have used the Jubula standalone version for experimentation.

- Used for black box testing. Supports code free testing.
- Can be used on Linux as well as Windows.
- Supports Java Toolkit - Java Swing, RCP, JavaFX and HTML applications.
- Has a detailed documentation that comes along with the Jubula Standalone installation. It is also the best resource available online.
- Fairly easy to use. Understanding the components of the ITE and their working takes some time but creating test suites after that is very easy.
- It is extensible. Custom actions and components can be added. It also provides API for users to write tests in Java code.[11]
- In case of any failure, automatic screenshot is taken, which can be seen by hovering over the failed test step. This helps in capturing the error.
- Highly modular. Test cases are built using test steps and Test suites are built using test cases. All of these components are reusable.
- The last version was released on 11 March 2019. Enjoys the support of Eclipse Foundation. The repository of Jubula Core on Github is a part of Eclipse Foundation. It has 8 contributors. [10]
- Inbuilt functionality to generate test reports is present.

- Can be integrated with Jenkins. Since it uses an embedded database by default to store the project, the Jenkins user does not have access to it. Simple workaround is to make the Jenkins user same as the user who created the project. Also xvfb plugin for Jenkins is required.

SikuliX

SikuliX automates anything that you see on your screen. It uses image recognition powered by OpenCV to identify GUI components. SikuliX comes with basic text recognition (OCR) and can be used to search text in images. [12]

- It is not Framework dependent. Can automate Web applications, Desktop applications and even Mobile applications if run on emulator.
- Requires Java installation as it is available as a JAR file.
- Supports Python(Jython), Ruby(JRuby), JavaScript and any Java Aware scripting language (Scala, Clojure) for writing scripts.[12]
- Supports Debugging.
- It is extensible. Extensions can be added using the IDE by clicking on Tools and then on extensions. Sikuli extensions can be Python/Jython modules and/or Java classes [19].
- Used for black box testing. Very easy to use.
- Detailed documentation available which is easy to follow.
- Overhead of images, all the images used in the test cases are stored in the system.
- Can be used on Linux, Windows and MacOS [12]
- Very easy to use.
- Integration with Jenkins is complex.

Selenium

- Can only be used for web applications. Similar tools like 'Winium' for Windows Desktop applications and 'Appium' for testing Android and IOS apps are available [15].
- It performs the minimum function of automating the browser or in simple words, communicating with the browser. This functionality is very useful for functional testing.
- Supports multiple programming languages for writing scripts - Java, Python, C, Ruby, JavaScript and Kotlin.
- The Selenium GitHub repository is well maintained and is currently being used by 54.5K users and has 502 contributors [17]. It clearly has a great community support.
- Can be used on Linux, Windows and macOS [18].
- Documentation is not detailed enough. Additional resources are required.
- Does not support codeless testing but is easy to learn and use.
- Selenium is not a complete Framework and it lacks in lot of things, but in can be integrated with a variety of other tools to overcome its shortcomings.
- It does not have test reporting capabilities but it can be integrated with TestNG to generate test reports.
- It does not have support for image testing but it can be integrated with SikuliX to achieve the same.
- It can be integrated with CI/CD tools like Jenkins.

JMeter

- The Apache JMeter is an open source software which is designed to load test functional behavior and measure performance. It is a 100 percent pure JAVA application

- It has very easy to follow documentation
- Complete portability and 100 percent Java purity.
- It has the ability to load and performance test many different applications, servers and protocols.
- JMeter also supports integration with Selenium, which allows it to run automation scripts alongside performance or load tests.
- Caching and offline analysis of test results.
- Many companies including DATACOM, Lufthansa, Sharpmind uses JMeter [20].

The experimental results are presented in a tabulated form below.

	Jnuit5	SikuliX	Jubula	Selenium	JMeter
Functionality	Unit testing & Integration testing	Functional Testing	GUI Functional Testing	Functional Testing	Performance Testing
Programming Language	Java	Jython, Jruby, JavaScript, Scala & Clojure	Codeless testing	Java, Python, C, Ruby, JavaScript and Kotlin	Codeless testing
Framework	Can only test Java Frameworks	Not Framework dependent. Can automate Web, Desktop and even Mobile apps, if run on emulator.	Java Toolkit - Java Swing, RCP, JavaFX and HTML applications	Can only be used to test Web Applications	Can be used to test Websites, Web Services, Database Servers, Shell scripts
OS support	Linux, Windows and MacOS	Linux, Windows and MacOS	Linux, Windows and MacOS	Linux, Windows and MacOS	Linux, Windows and MacOS
Learning Curve	Smooth learning curve, if familiar with Java.	Easy to learn	Mild learning curve	Steep learning curve	Low learning curve
Community Support (5 being the highest)	5	2	3	5	4
Extensible	Can be extended using the Extensions API	Extensible. Sikuli extensions can be Python/Jython modules and/or Java classes.	Toolkit extensions for custom UI widgets can be created. Provides API for users to write tests in Java.	Highly extensible. Can be easily integrated with other open source tools to enhance its functionality.	Highly extensible, Can support many third-party apps and plug-ins.
Documentation	Easy to follow Documentation	Easy to follow, good for beginners	Well Detailed Documentation	Documentation is not detailed enough. Additional resources required.	Well Detailed Documentation
Inbuilt Reports Generation	Yes	No	Yes	No	Yes
Integration with Jenkins	Easy	Complex	Moderate	Easy	Easy

Conclusion

It can be very overwhelming to choose from a plethora of open source automation testing tools. There is no perfect testing tool that can be used for all softwares. It is very important to understand your software's requirements thoroughly and then compare amongst various tools. The ROI of automation testing is highly dependent on one's choice of the testing automation tool. Choosing a wrong testing tool can prove very expensive. Good amount of time must be spent on selecting the perfect fit automation testing tool for one's software.

References

- [1] <http://elementalselenium.com/tips/38-headless>
- [2] <https://docs.datastax.com/en/jdk-install/doc/jdk-install/installOpenJdkDeb.html>
- [3] <https://www.journaldev.com/21711/junit-setup-maven>
- [4] <https://www.vogella.com/tutorials/JUnit/article.html>
- [5] <https://junit.org/junit5/docs/current/user-guide/>
- [6] <https://junit.org/junit5/docs/current/user-guide/#extensions>
- [7] <https://www.eclemma.org/jacoco/>
- [8] <https://discovery.hgdata.com/product/junit>
- [9] <https://github.com/junit-team/junit5>
- [10] <https://github.com/eclipse/jubula.core>
- [11] <https://testing.bredex.de/feature-overview.html>
- [12] <http://sikulix.com/>
- [13] <https://chromedriver.chromium.org/getting-started>
- [14] <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>
- [15] https://www.rapidvaluesolutions.com/tech_blog/test-automation-winium/
- [16] <https://www.selenium.dev/documentation/en/>
- [17] <https://github.com/SeleniumHQ/selenium>
- [18] https://www.selenium.dev/documentation/en/webdriver/driver_requirements/
- [19] https://sikulix-2014.readthedocs.io/en/latest/extensions/extensions_info.html
- [20] <https://cwiki.apache.org/confluence/display/JMETER/JMeterUsers>
- [21] <https://jmeter.apache.org/>
- [22] <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-16-04>