Maven & Jenkins with Selenium: Complete Tutorial

What is Jenkins?

Jenkins is the leading open-source continuous integration tool developed by Hudson lab. It is cross-platform and can be used on Windows, Linux, Mac OS and Solaris environments. Jenkins is written in Java. Jenkin's chief usage is to monitor any job which can be SVN checkout, cron or any application states. It fires pre-configured actions when a particular step occurs in jobs.

In this tutorial, we will learn

Important Features of Jenkins

Important Features of Jenkins

- Why Jenkins and Selenium?
- Steps to use install Maven and use it with TestNG Selenium
- Steps to Install Jenkins and configure it to Run Maven with TestNg Selenium
- Scheduling Jenkins for automatic execution.
- Jenkins with TestNg
- Benefits of Jenkins

Important Features of Jenkins

- Change Support: Jenkins generates the list of all changes done in repositories like SVN.

- Permanent links: Jenkins provides direct links to the latest build or failed build that can be used for easy communication
- Installation: Jenkins is easy to install either using direct installation file (exe) or war file to deploy using application server.
- Email integration: Jenkins can be configured to email the content of the status of the build.
- Easy Configuration: To configure various tasks on Jenkins is easy.
- TestNG test: Jenkins can be configured to run the automation test build on TestNG after each build of SVN.
- Multiple VMs: Jenkins can be configured to distribute the build on multiple machines.
- Project build: Jenkins documents the details of jar, version of jar and mapping of build and jar numbers.
- Plugins: 3rd party plugin can be configured in Jenkins to use features and additional functionality.

Why Jenkins and Selenium?

- Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass.
- Jenkins can schedule your tests to run at specific time.
- You can save the execution history and Test Reports.
- Jenkins supports Maven for building and Testing a project in continuous integration.

Why Maven & Jenkins

Selenium WebDriver is great for browser automation. But, when using it for testing and building a test framework, it feels underpowered. Integrating Maven with Selenium provides following benefits
Apache Maven provides support for managing the full lifecycle of a test project.

- Maven is used to define project structure, dependencies, build, and test management.
- Using pom.xml(Maven) you can configure dependencies needed for building testing and running code.
- Maven automatically downloads the necessary files from the repository while building the project.

Steps to use install Maven and use it with TestNG Selenium

For this tutorial, we will use Eclipse (Juno) IDE for Java Developers to set up Selenium WebDriver Project. Additionally, we need add m2eclipse plugin to Eclipse to facilitate the build process and create pom.xml file. Let's add m2eclipse plugin to Eclipse with following steps:

Step1) In Eclipse IDE, select **Help** | **Install New Software** from Eclipse Main Menu.

**Step 2)** On the Install dialog, select **Work with** and m2e plugin as shown in the following screenshot:
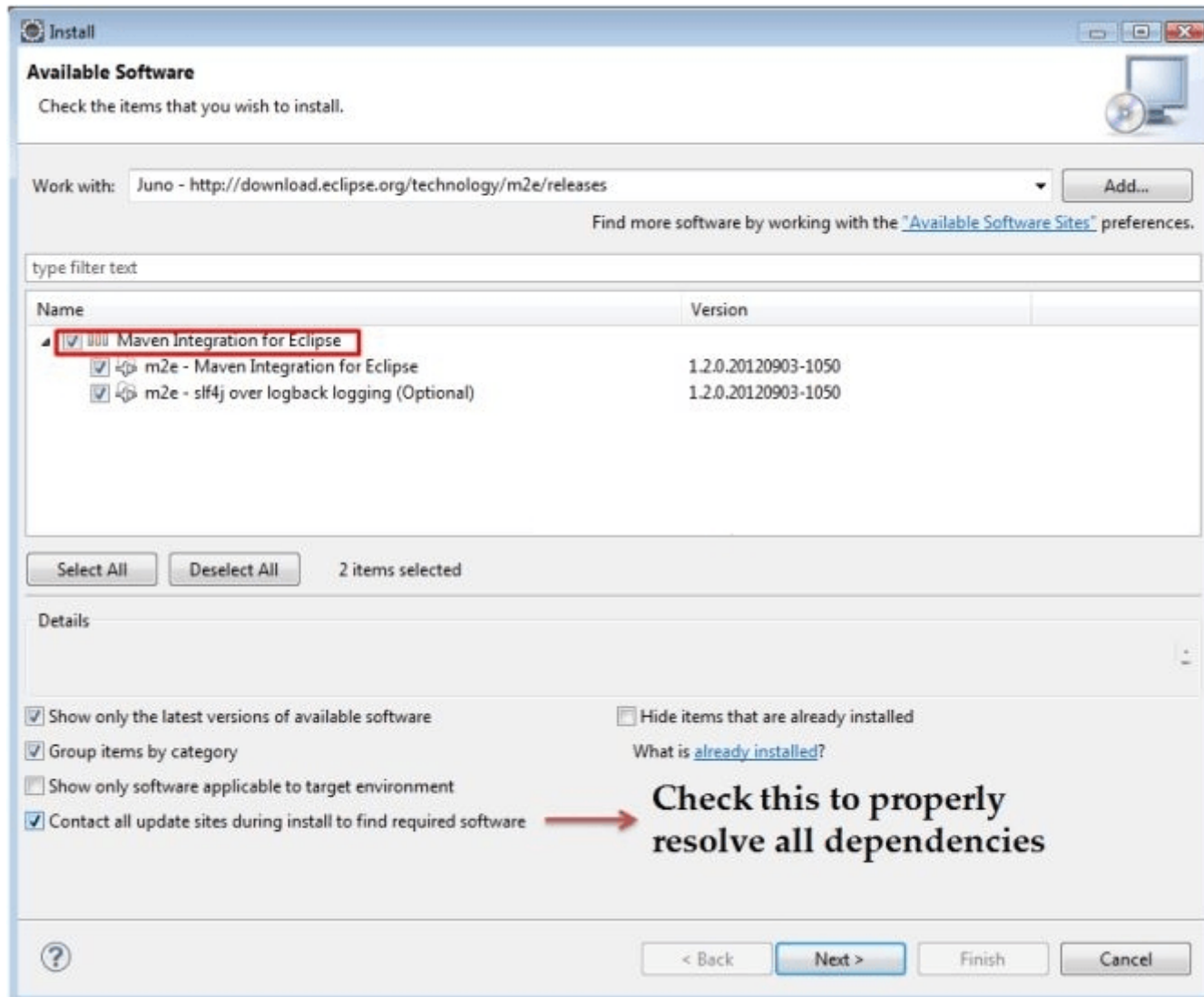
Step 3) Click on **Next** button and finish installation.

Configure Eclipse with Maven

With m2e plugin is installed, we now need create Maven project.

Step 1) In Eclipse IDE, create a new project by selecting **File** | **New** | **Other** from Eclipse menu.

Step 2) On the **New** dialog, select **Maven** | **Maven Project** and click Next
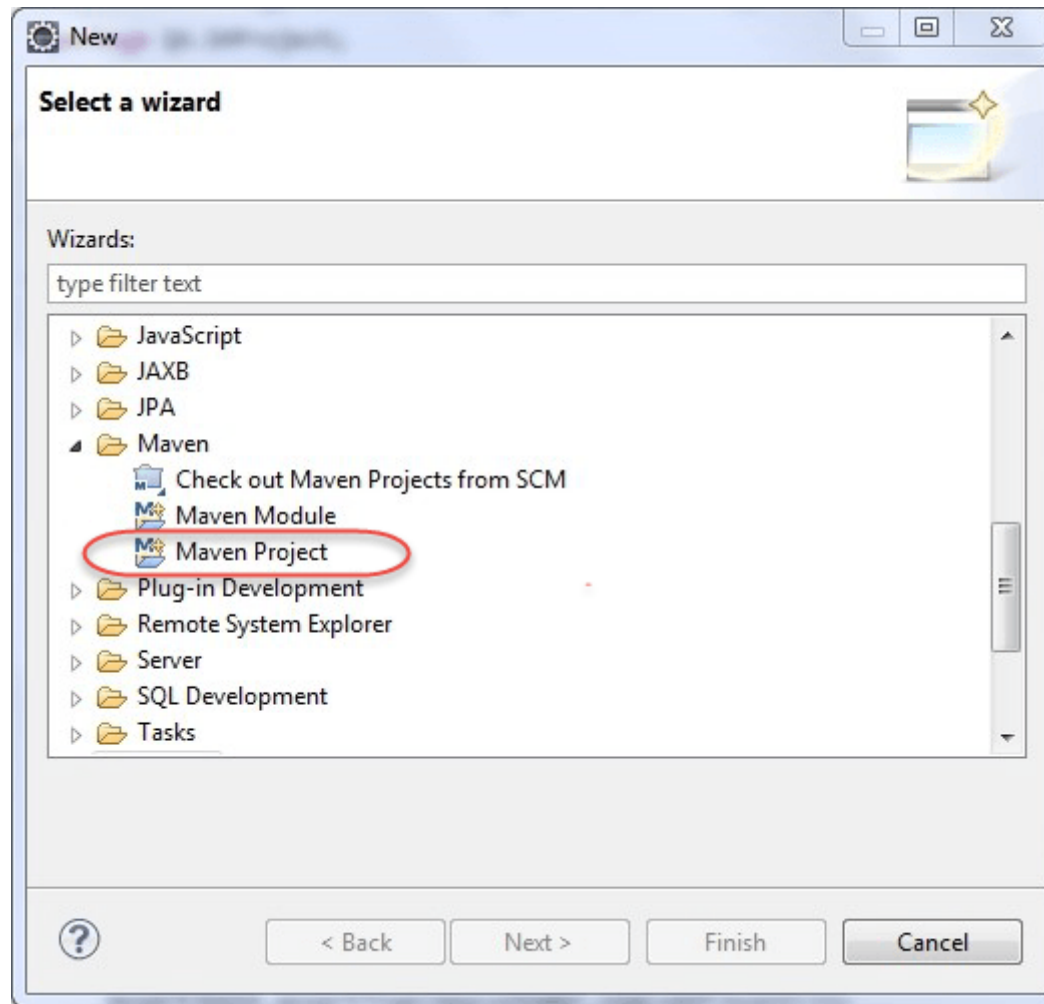
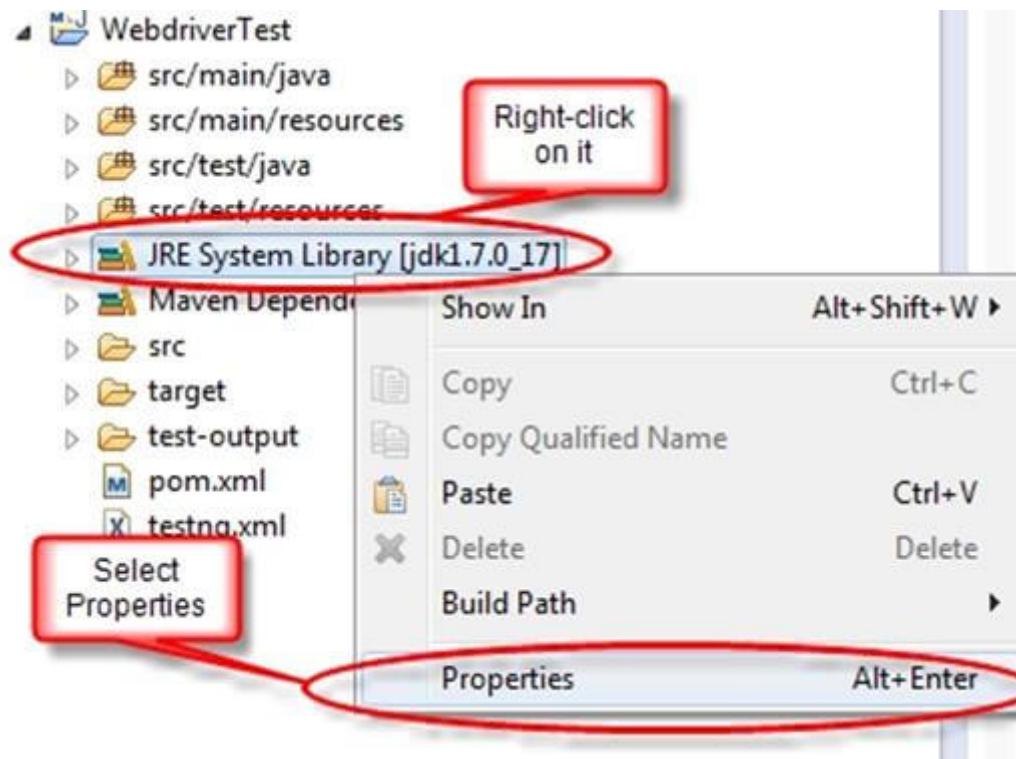**Step 3)** On the **New Maven Project** dialog select the **Create a simple project** and click Next
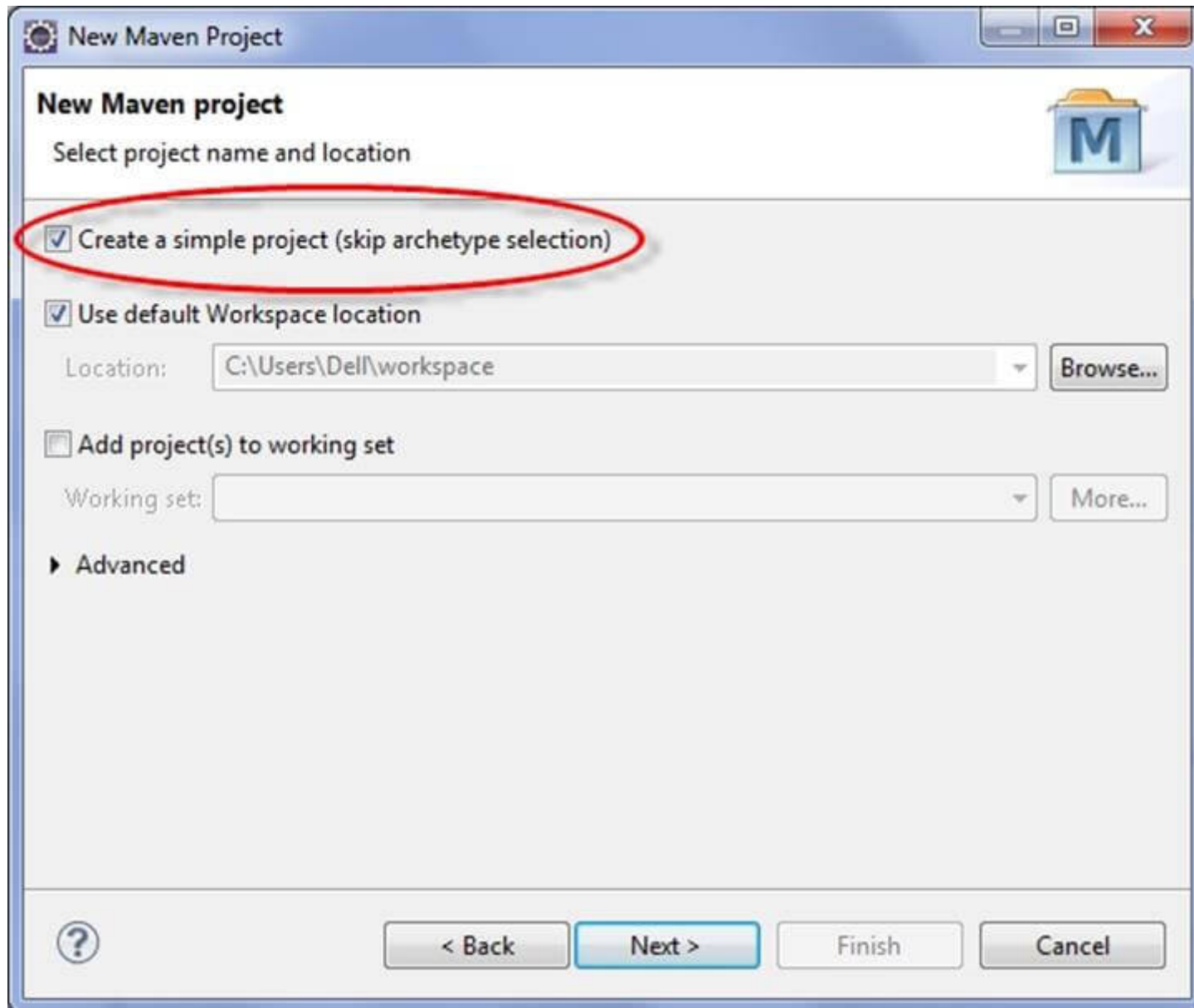
Step 4) Enter WebdriverTest in **Group Id**: and **Artifact Id**: and click finish
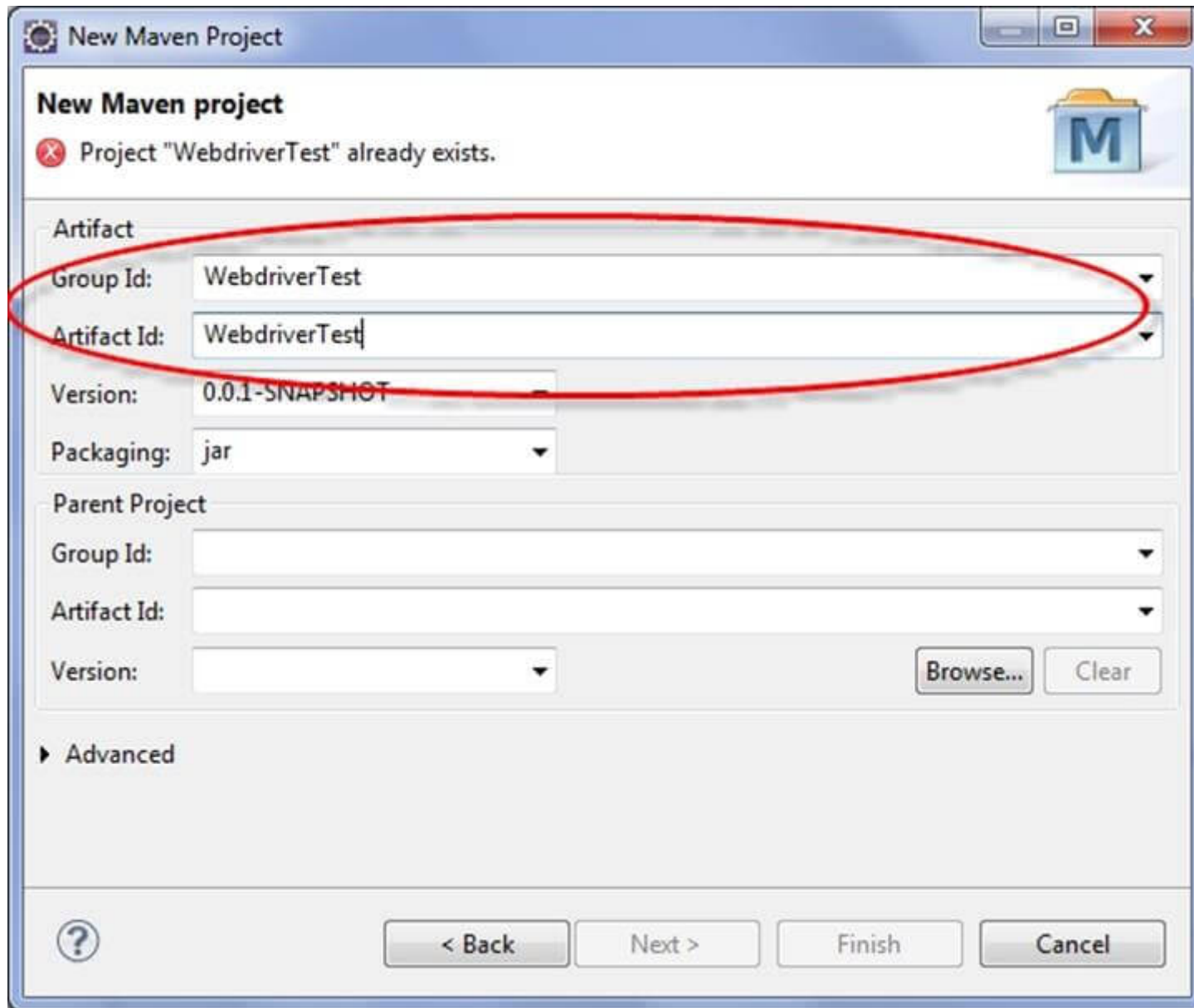
Step 5) Eclipse will create **WebdriverTest** with following structure:
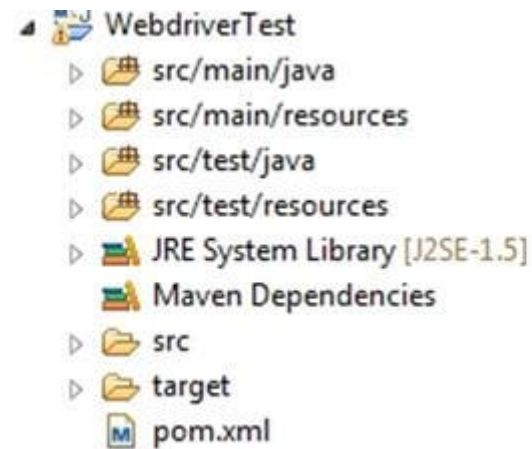
Step 6) Right-click on **JRE System Library** and select the **Properties** option from the menu.
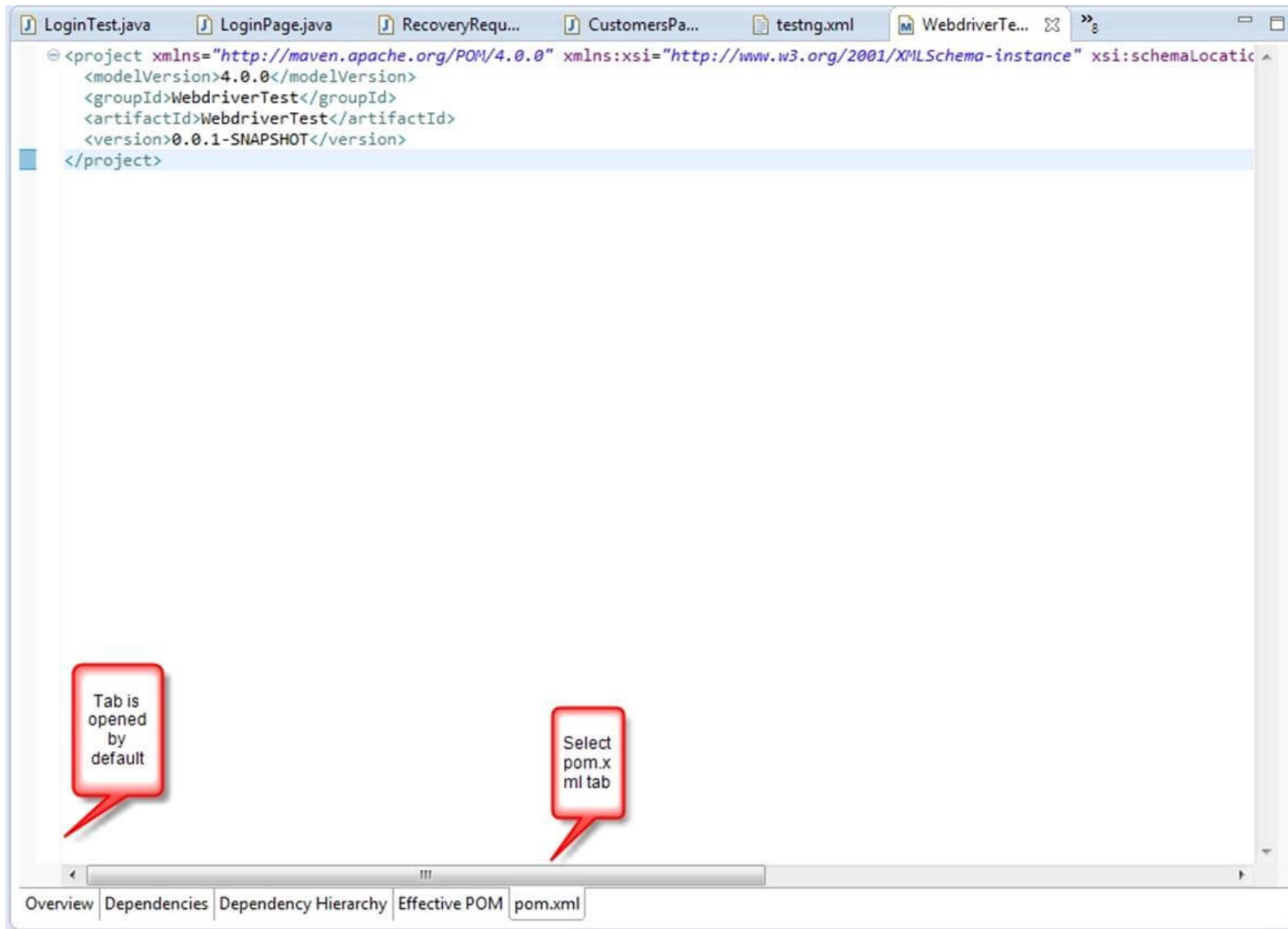
On the **Properties for JRE System Library** dialog box, make sure **Workspace default JRE** is selected and click OK
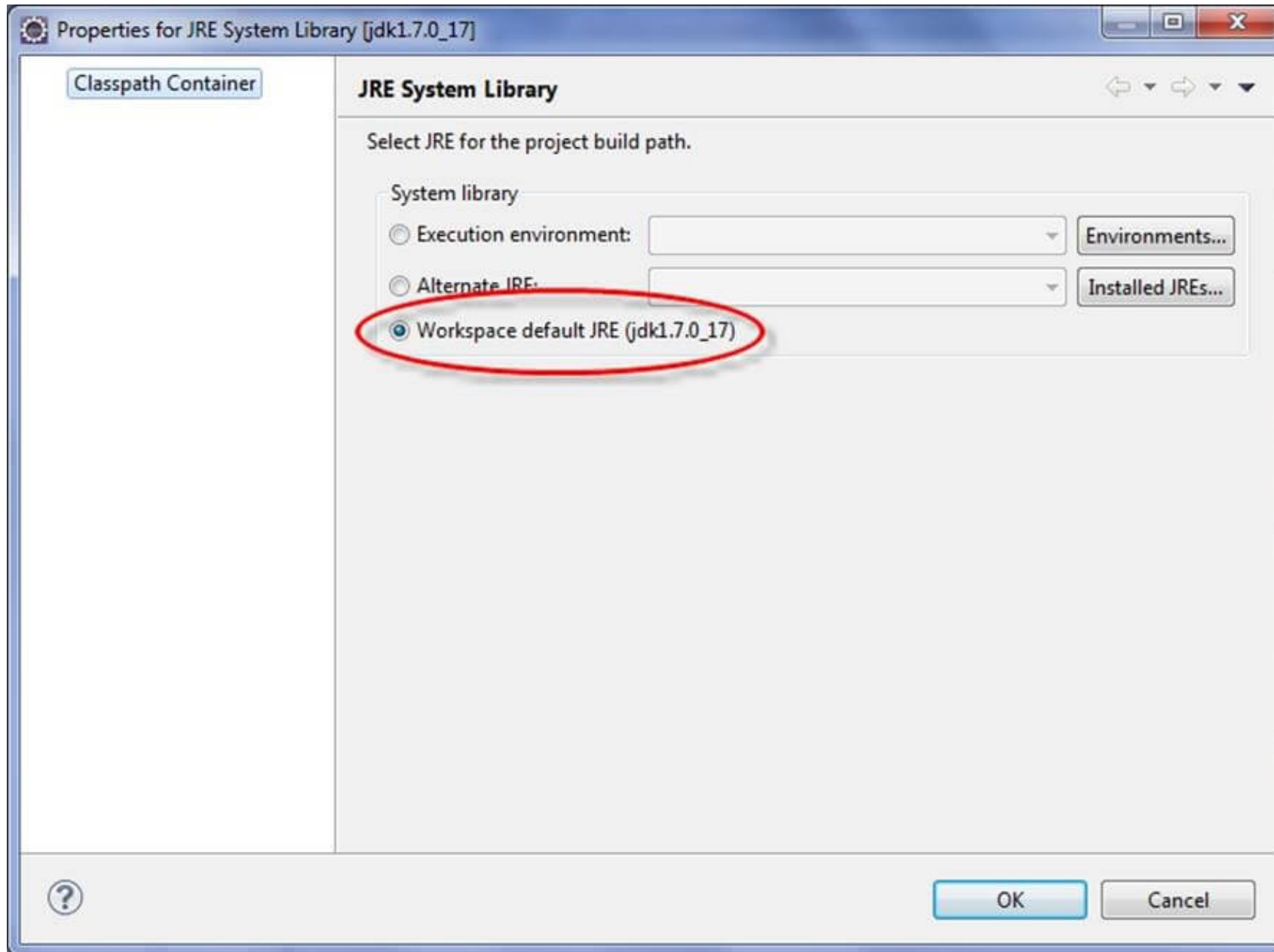
Step 7). Select **pom.xml** from **Project Explorer**..
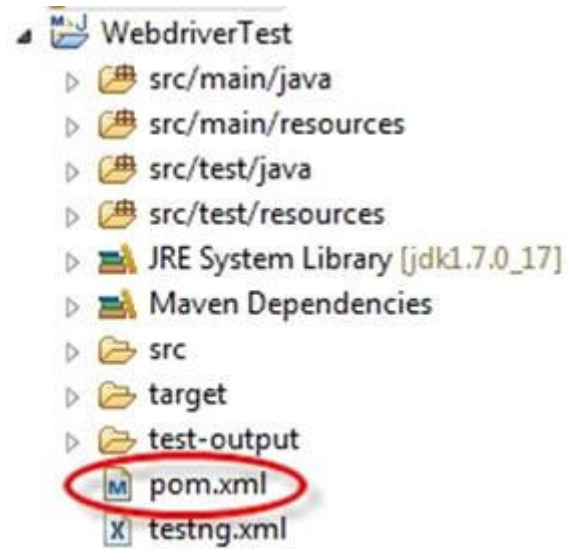
pom.xml file will Open in Editor section

Step 8). Add the Selenium and TestNG, JUnit dependencies to pom.xml in the <project> node:

```
<dependencies>
<dependency>
```

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.45.0</version>
                </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.8</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

Step 9) Create a New TestNG Class. Enter Package name as "example" and "NewTest" in the **Name**: textbox and click on the **Finish** button as shown in the following screenshot:

New TestNG class

Specify additional information about the test class.

Source folder: /WebdriverTest/src/test   Browse...

Package name: example   Browse...

Class name: NewTest

Annotations
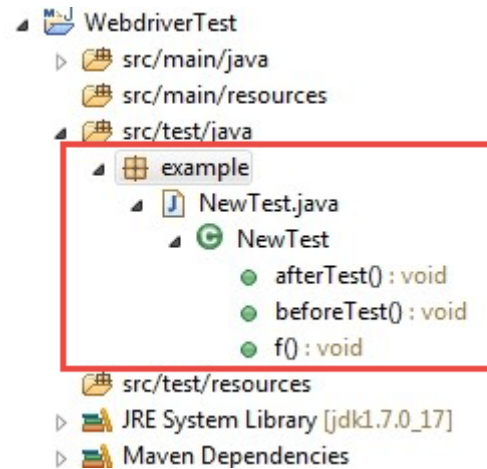
☐ @BeforeMethod   ☐ @AfterMethod   ☐ @DataProvider
☐ @BeforeClass   ☐ @AfterClass
☑ @BeforeTest   ☑ @AfterTest
☐ @BeforeSuite   ☐ @AfterSuite

XML suite file:

⟨?⟩   < Back   Next >   Finish   Cancel

Step 10). Eclipse will create the NewTest class as shown in the following screenshot:

Step 11) Add the following code to the **NewTest** class:

This code will verify the title of Guru99 Selenium Page

```
package example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.AfterTest;
public class NewTest {
          private WebDriver driver;
              @Test
              public void testEasy() {
                      driver.get("http://www.guru99.com/selenium-tutorial.html");
                      String title = driver.getTitle();
                      Assert.assertTrue(title.contains("Free Selenium Tutorials"));
              }
              @BeforeTest
              public void beforeTest() {
```
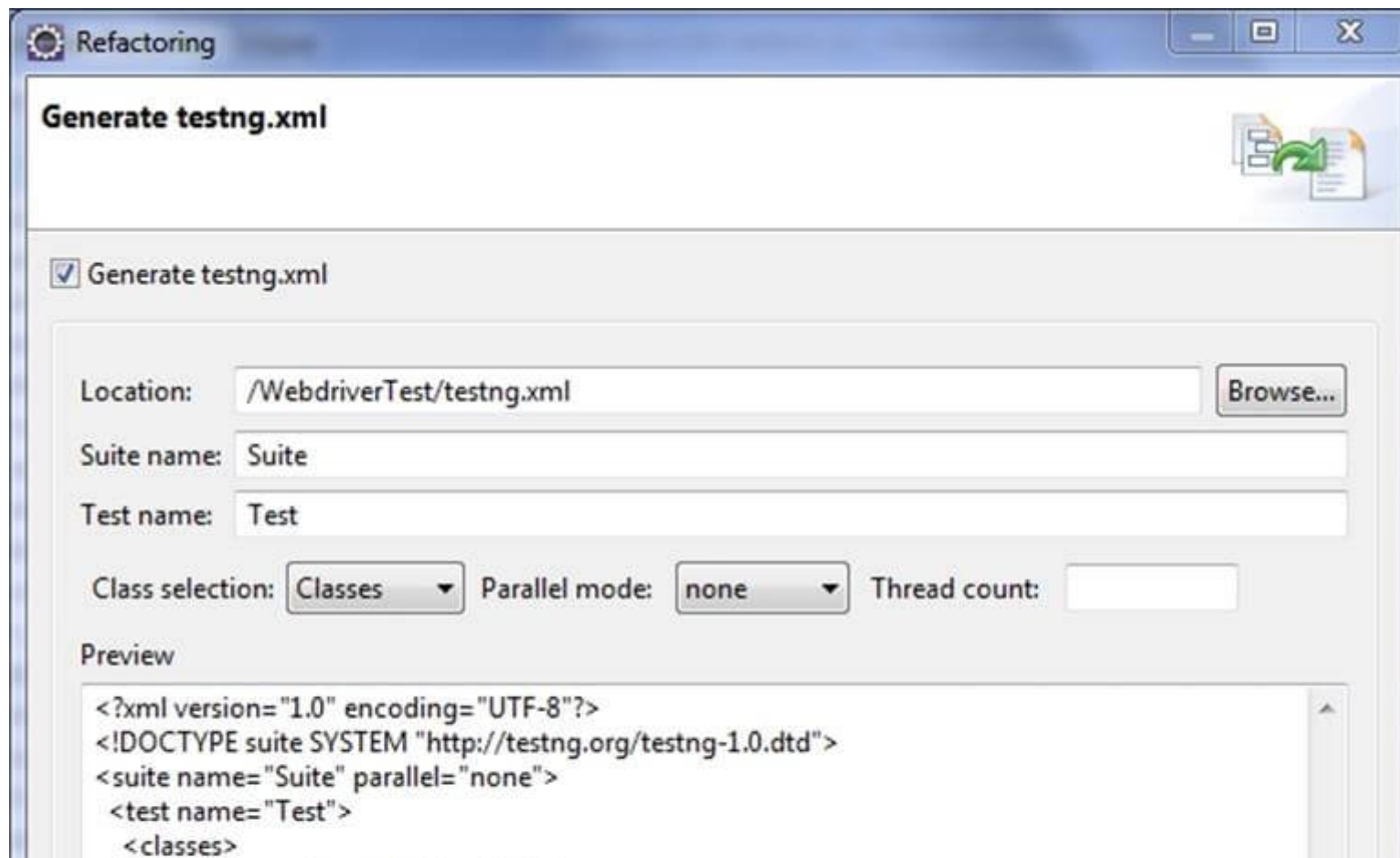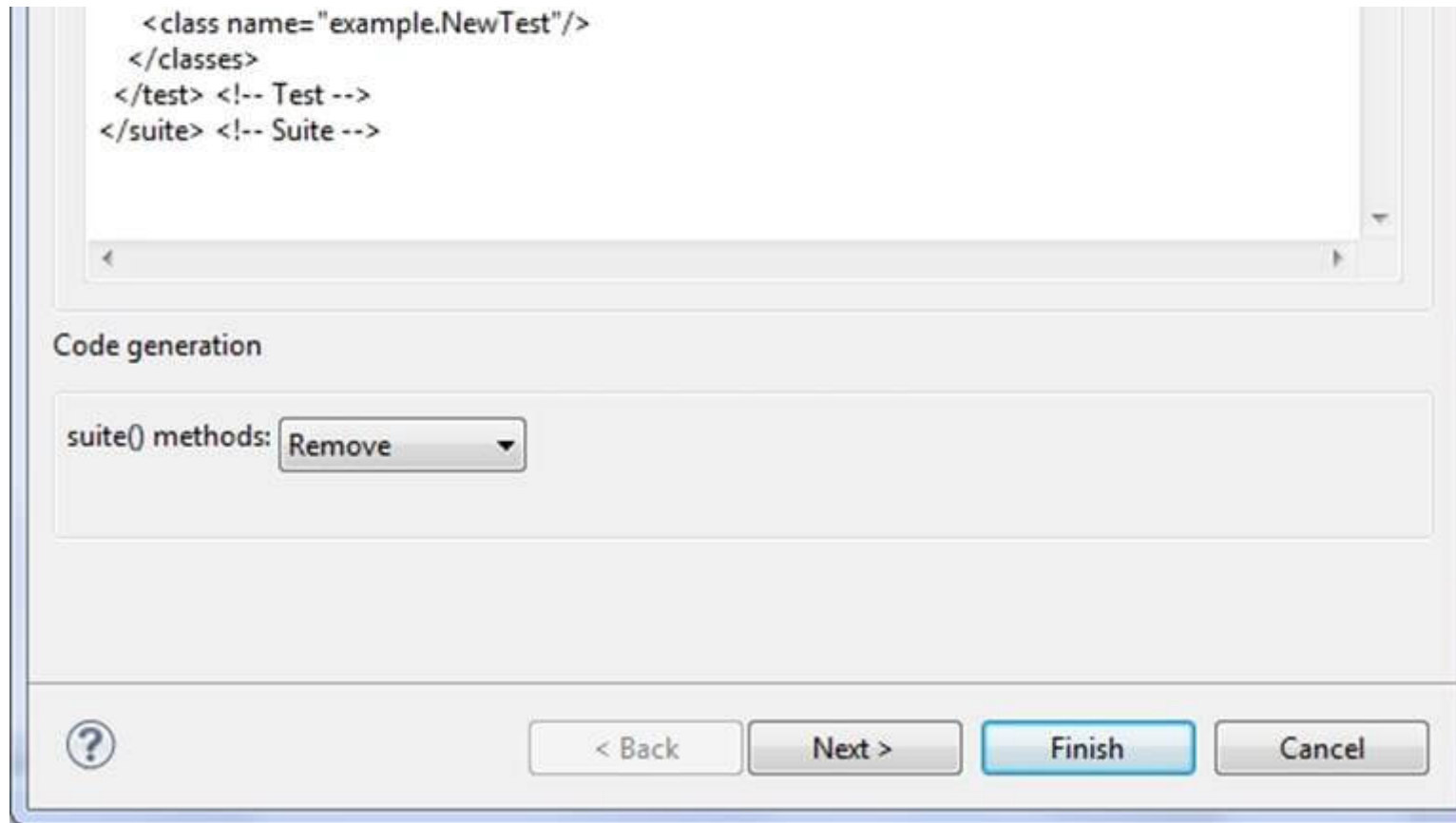
```
            driver = new FirefoxDriver();
        }
        @AfterTest
        public void afterTest() {
                driver.quit();
        }
}
```

Step 12) Right-click on the WebdriverTest and select **TestNG | Convert to TestNG**.
Eclipse will create testng.xml which says that you need to run only one test with the name **NewTest** as shown in the following screenshot:
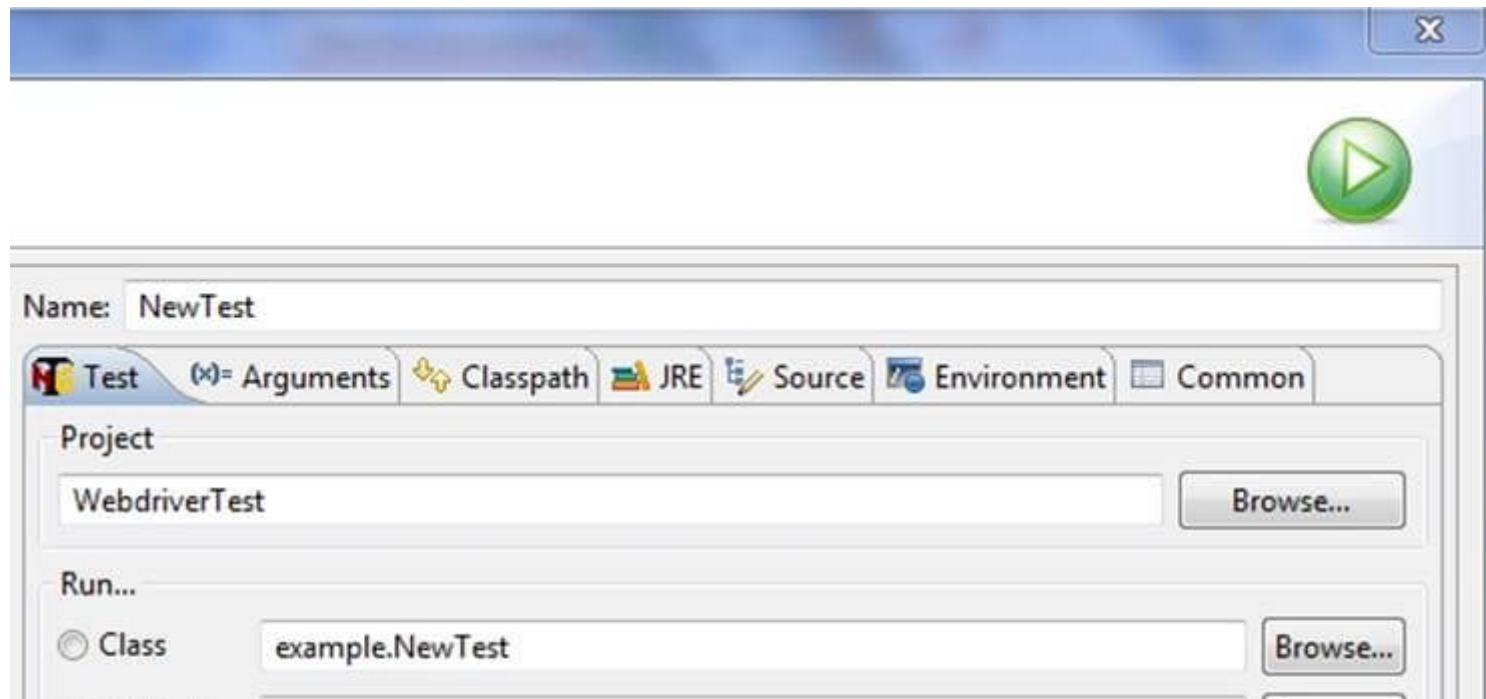
```
        <class name="example.NewTest"/>
      </classes>
    </test> <!-- Test -->
  </suite> <!-- Suite -->
```

Code generation

suite() methods: Remove ▼

(?)   < Back   Next >   Finish   Cancel

Update the project and make sure that file appears in the tree **Package Explorer** (right click on the project -
Refresh).
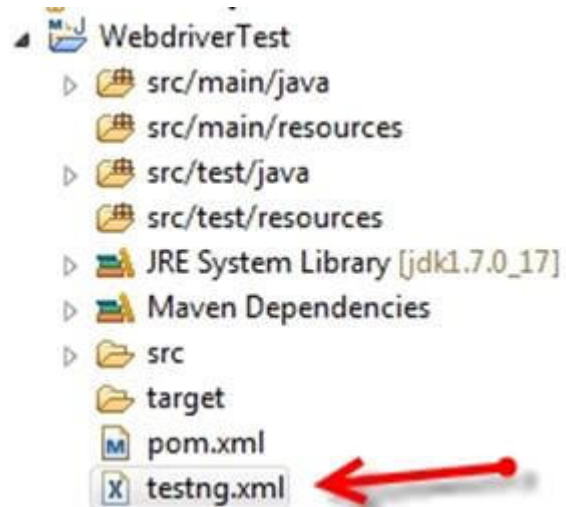
Step 13) Now you need to run test through this **testng.xml.**

So, go to the **Run Configurations** and create a new launch **TestNG**, select the project and field **Suite** as **testng.xml** and click Run

Make sure that build finished successfully.

Step 14). Additionally, we need to add

1. maven-compiler-plugin
2. maven-surefire-plugin
3. testng.xml

to pom.xml.

The maven-surefire-plugin is used to configure and execute tests. Here plugin is used to configure the testing.xml for TestNG test and generate test reports.

The maven-compiler-plugin is used to help in compiling the code and using the particular JDK version for compilation. Add all dependencies in the following code snippet, to pom.xml in the <plugin> node:

```
17      <plugins>
18        <plugin>
19        <groupId>org.apache.maven.plugins</groupId>
20          <artifactId>maven-compiler-plugin</artifactId>
21          <version>2.3.2</version>
22          <configuration>
23            <source>1.7</source>
24            <target>1.7</target>
25          </configuration>
26        </plugin>
27        <plugin>
28            <groupId>org.apache.maven.plugins</groupId>
29            <artifactId>maven-surefire-plugin</artifactId>
30            <version>2.12</version>
31            <inherited>true</inherited>
32              <configuration>
33            <suiteXmlFiles>
34              <suiteXmlFile>testng.xml</suiteXmlFile>
35            </suiteXmlFiles>
36          </configuration>
37        </plugin>
38      </plugins>
```

Step 15) To run the tests in the Maven lifecycle, Right-click on the WebdriverTest and select **Run As | Maven test**. Maven will execute test from the project.
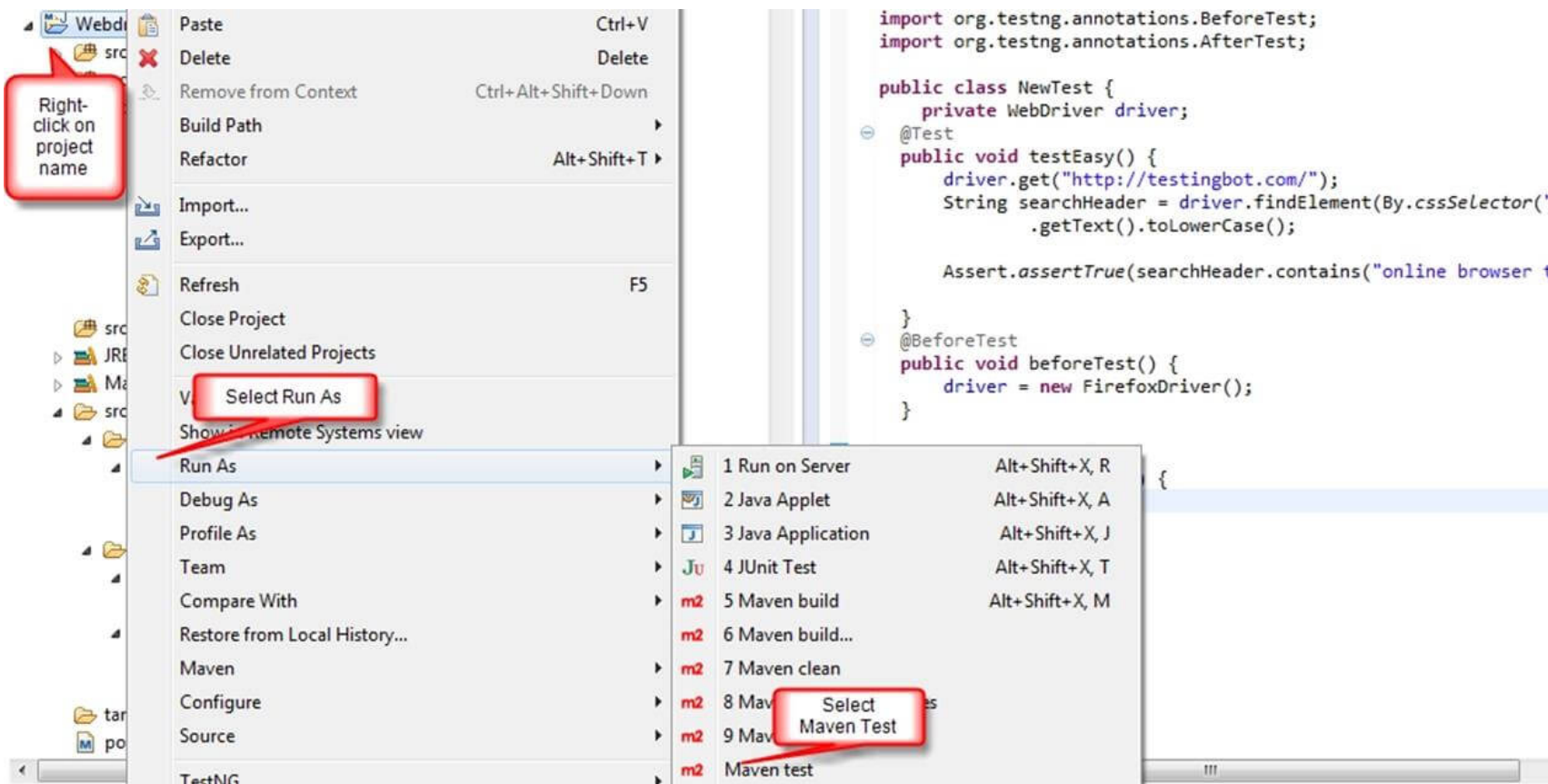
Make sure that build finished successfully.

Steps to Install Jenkins and configure it to Run Maven with TestNg Selenium

Installation

Step 1) Go to http://jenkins-ci.org/and download correct package for your OS. Install Jenkins.

Download Jenkins

Java Web Archive (.war)

Latest and greatest (1.607)

changelog | past releases | RC

upgrading from Hudson?

Or native package

Windows

Ubuntu/Debian

Red Hat/Fedora/CentOS

Mac OS X

openSUSE

FreeBSD

OpenBSD
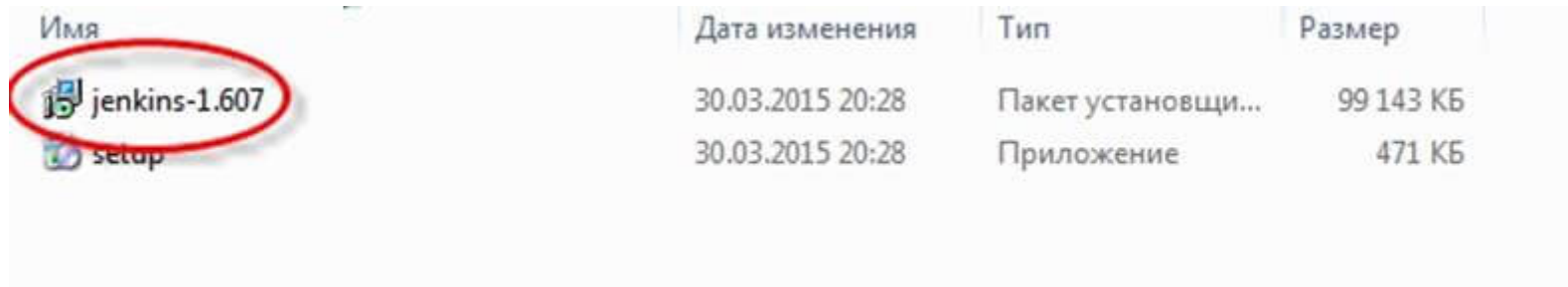
Solaris/OpenIndiana

Gentoo

Step 2) Unzip Jenkins to specified folder. Run exe file as shown in following screenshot:

Имя	Дата изменения	Тип	Размер
jenkins-1.607	30.03.2015 20:28	Пакет установщи...	99 143 КБ
setup	30.03.2015 20:28	Приложение	471 КБ

Step 3) In **Jenkins 1.607 Setup** window click on **Next** button.

Step 4) Click on **Install** button in the end.
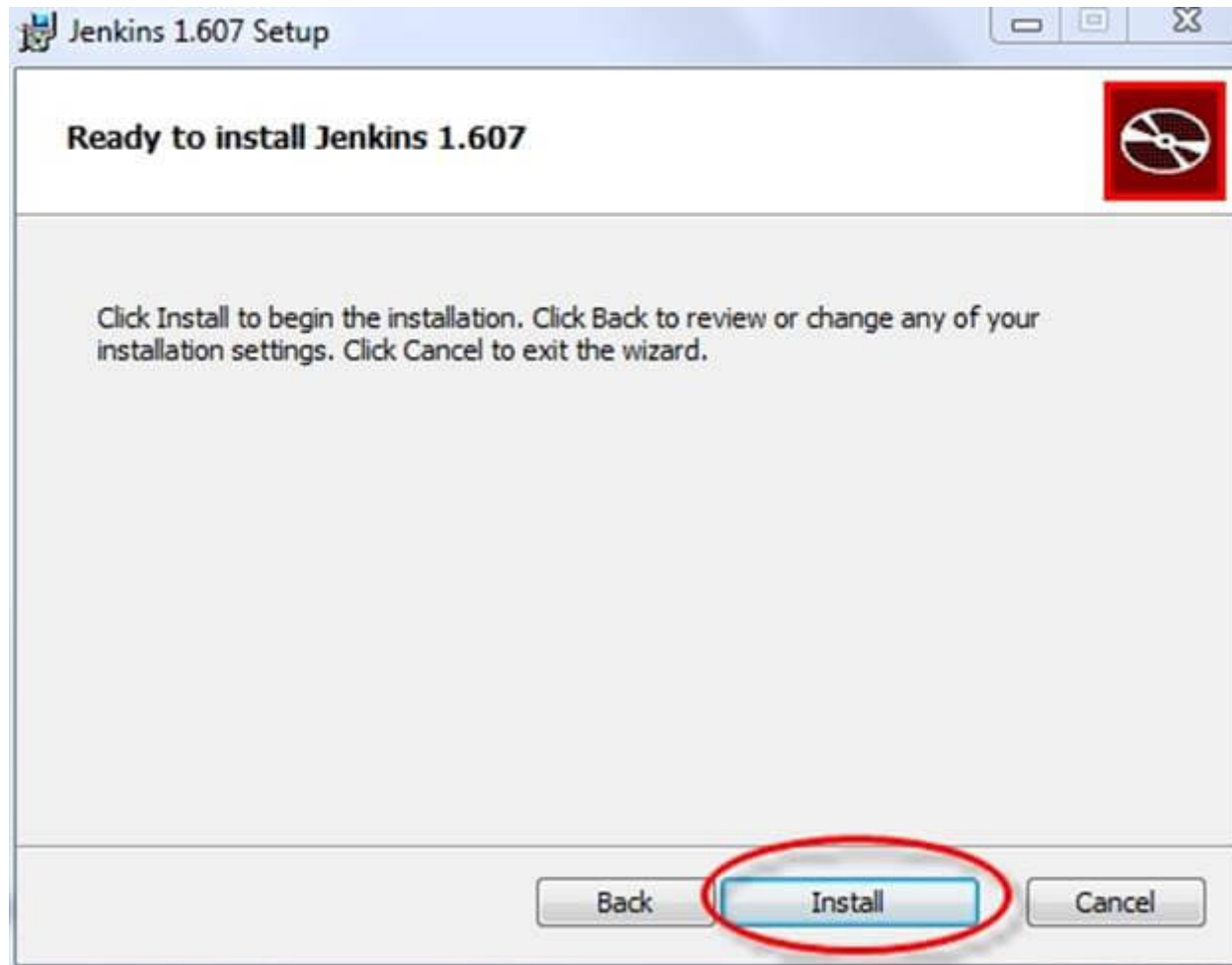
Step 5) Once installation is done, navigate to the Jenkins Dashboard (http://localhost:8080 by default) in the browser window.
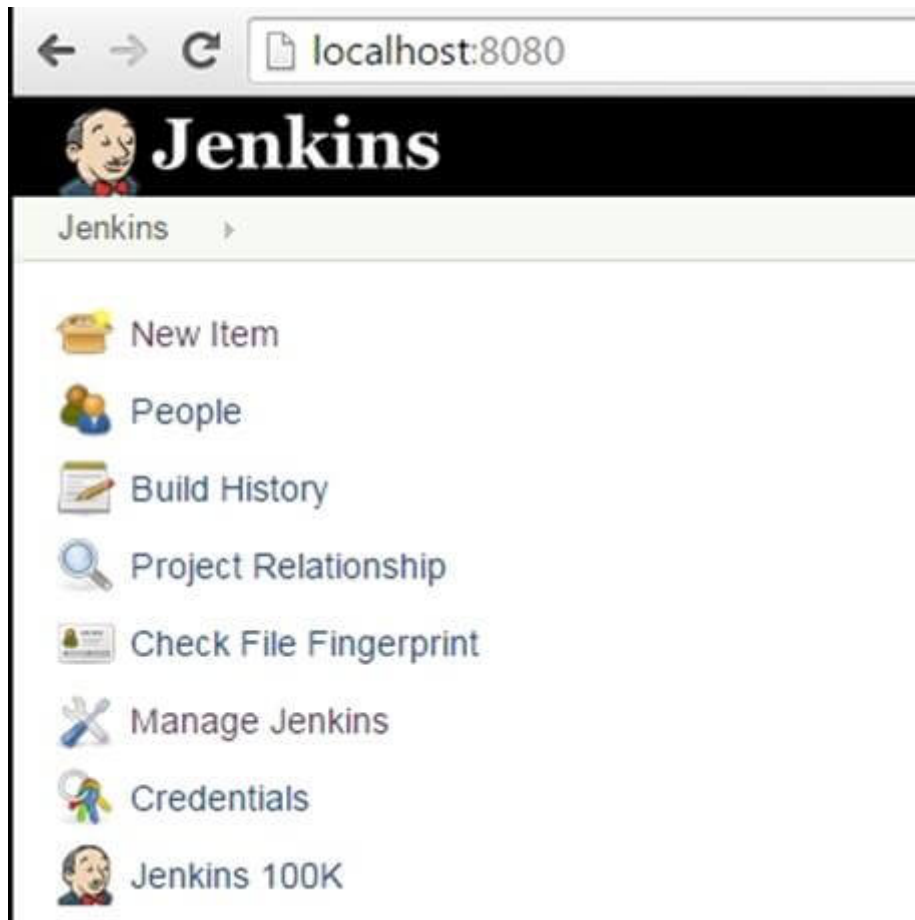
Step 6) Click on the **New Item** link to create a CI job.

Step 7) Select the Maven project radio button as shown in the following screenshot:

Item name	WebdriverTest

○ **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, an something other than software build.

⦿ **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

○ **External Job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. Th Jenkins as a dashboard of your existing automation system. See the documentation for more details.

○ **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments,

○ **Copy existing Item**
Copy from [   ]

OK

Using the Build a **Maven Project** option, Jenkins supports building and testing Maven projects.

Step 6) Click on OK button. A new job with name "WebdriverTest" is created in Jenkins Dashboard.

WebdriverTest   N/A   N/A

Step 7) Go to **Manage Jenkins** => **Configure System** as shown in the following screenshot.

Click on JDK installations and configure JDK as in the following screenshot:

Step 8) Go to the **Build** section of new job.

- In the **Root POM** textbox, enter full path to pom.xml
- In Goals and options section, enter "clean test"

Step 9) Click on **Apply** button.

Step 10) On the WebdriverTest project page, click on the **Build Now** link.

Maven will build the project. It will then have TestNG execute the test cases.

Step 11) Once the build process is completed, in Jenkins Dashboard click on the **WebdriverTest** project

Icon: S M L

Step 12) The WebdriverTest project page displays the build history and links to the results as shown in the following screenshot:

⬤ #8	Mar 31, 2015 11:48 PM
⬤ #7	Mar 31, 2015 11:45 PM
⬤ #6	Mar 31, 2015 3:17 PM
⬤ #5	Mar 31, 2015 12:12 PM
⬤ #4	Mar 30, 2015 12:53 AM
⬤ #3	Mar 30, 2015 12:42 AM
⬤ #2	Mar 29, 2015 11:17 PM
⬤ #1	Mar 29, 2015 9:26 PM

Permalinks

- Last build (#9), 19 hr ago
- Last stable build (#9), 19 hr ago
- Last successful build (#9), 19 hr ago
- Last failed build (#7), 19 hr ago
- Last unstable build (#8), 19 hr ago
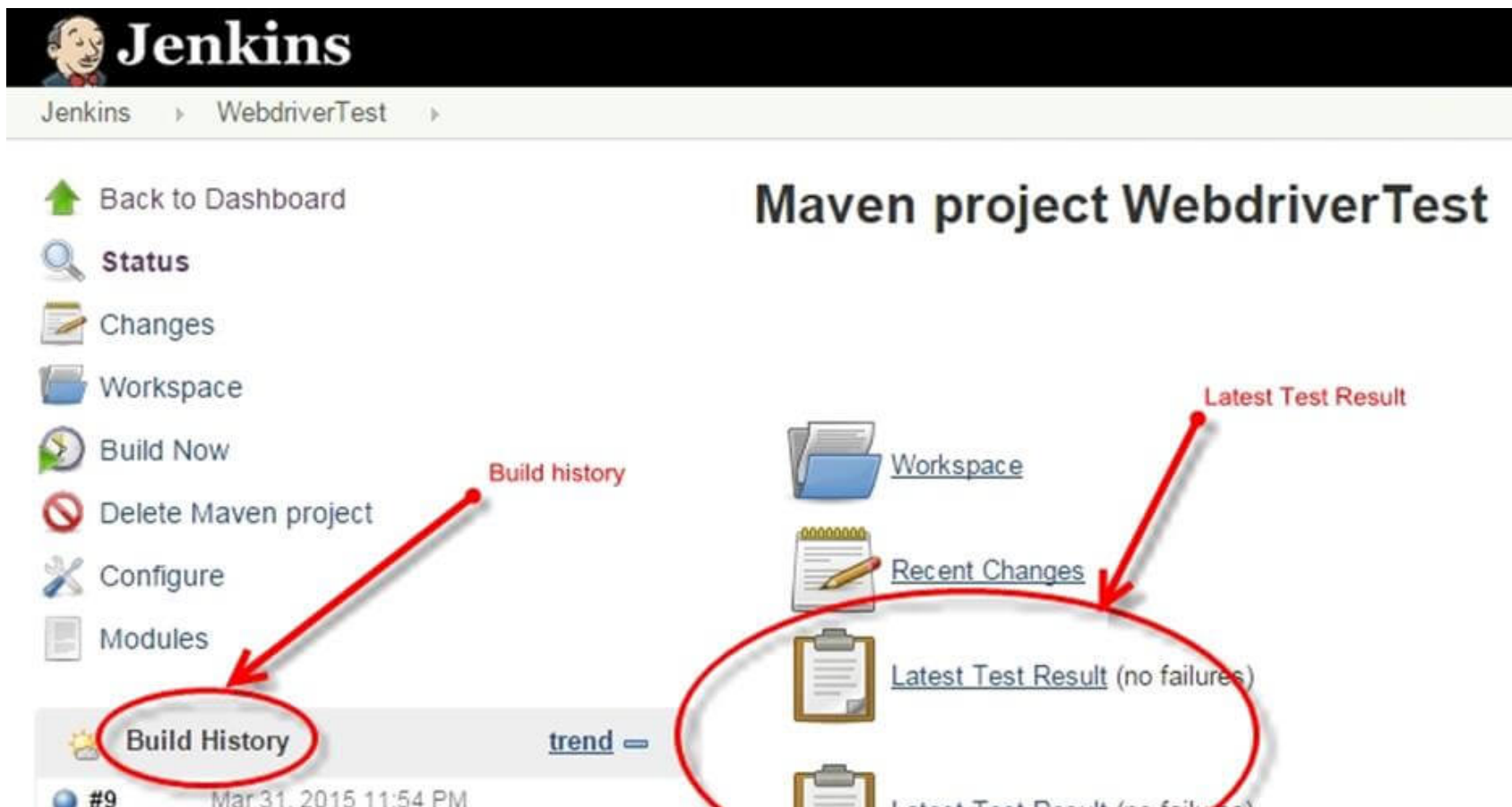- Last unsuccessful build (#8), 19 hr ago

RSS for all RSS for failures

Step 13) Click on the "Latest Test Result" link to view the test results as shown in the following screenshot:

Test Result

0 failures (-1)

6 tests (±0)

Module	Fail	(diff)	Total	(diff)
ToolsQA:DemoMavenProject	0	-1	6	

Step 14). Select specific build, and you will see the current status by clicking on "**console output**".

```
-------------------------------------------------------
  T E S T S
-------------------------------------------------------
Running TestSuite
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.748 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[JENKINS] Recording test results
log4j:WARN No appenders could be found for logger (org.apache.commons.beanutils.converters.BooleanConverter).
log4j:WARN Please initialize the log4j system properly.
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - --------------------------------------------------
--------------------------
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - BUILD SUCCESS
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - --------------------------------------------------
--------------------------
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Total time: 01:26 min
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Finished at: 2015-03-29T21:28:51+03:00
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - Final Memory: 25M/52M
[pool-1-thread-1 for channel] INFO org.apache.maven.cli.event.ExecutionEventLogger - --------------------------------------------------
--------------------------
Ожидаю пока Jenkins закончит сбор данных
[JENKINS] Archiving C:\Users\Dell\workspace\FirstWebdriverTest\pom.xml to ToolsQA/DemoMavenProject/0.0.1-SNAPSHOT/DemoMavenProject-
0.0.1-SNAPSHOT.pom
channel stopped
Finished: SUCCESS
```

Scheduling Jenkins for automatic execution.

Scheduling builds(Selenium Tests) is one of the important features of Jenkins where it automatically triggers the build, based on defined criteria. Jenkins provides multiple ways to trigger the build process under the Build Trigger configuration.

For example:
Enter 0 23 * * * in the Schedule textbox as shown in the following screenshot. This will trigger the build process every day at 11 p.m.

Using Jenkings without Maven

To run pure TestNg script in Jenkins, enter the following in build

D:>java -cp "Pathtolibfolder\lib*;Pathtobinfolder\bin" org.testng.TestNG testng.xml

Build

⸬ **Execute Windows batch command**   ⑦

Command D:>java -cp "Pathtolibfolder\lib*;Pathtobinfolder\bin" org.testng.TestNG testng.xml|

See the list of available environment variables

Delete

Add build step ▼

Post-build Actions

Add post-build action ▼

Save   **Apply**

- Click on Save button.
- Note: The actual path of lib and bin folder need to add in above command.
- After saving the command, Jenkins will build project in predefined time, and this command will be run using TestNG.
- Result will be stored in custom report HTML file that can be sent via email with a Jenkin configuration
- Output of the code will be

Test results
1 suite

All suites	

Default suite

Info
- C:\Users\hiteshkumar_panchani\AppData\Local\Temp\testng-eclipse-1972880750\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 1 method, 1 passed
- Passed methods (hide) (show)
  - ☑ LiveLink105VM13

☑ temp.CopyOftestng1

LiveLink105VM13

C:\Users\hiteshkumar_panchani\AppData\Local\Temp\testng-eclipse-1972880750\testng-customsuite.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Default suite">
  <test verbose="2" name="Default test">
    <classes>
      <class name="temp.CopyOftestng1"/>
    </classes>
  </test> <!-- Default test -->
</suite> <!-- Default suite -->
```

Tests for Default suite

Default test (1 class)

Groups for Default suite

Times for Default suite

Benefits of using Jenkins

1. Early issue finding – Bug can be detected in early phase of the software development
2. Automatic integration – no separate effort required to integrate all changes
3. Installer – a deployable system available at any point of development
4. Records – part build records maintained
5. Support and Plugins: One of the reasons for Jenkin's popularity is the availability of large community support. Also, lots of ready-made plugins are available which help you expand its functionality.