

Maven



What is Maven?



How can maven benefit to the development process



Maven Architecture



Features of Maven



Maven Lifecycle



Mithilesh Singh

Struggle without maven?



While creating the project we always need 3rd party tools as dependency so have to install all those as jar files in the project manually.

If there is an update available, we need to upgrade the version of the jar files. So manually we need to delete the older version and download the new one. This is very hectic and time consuming process.

Why Maven?

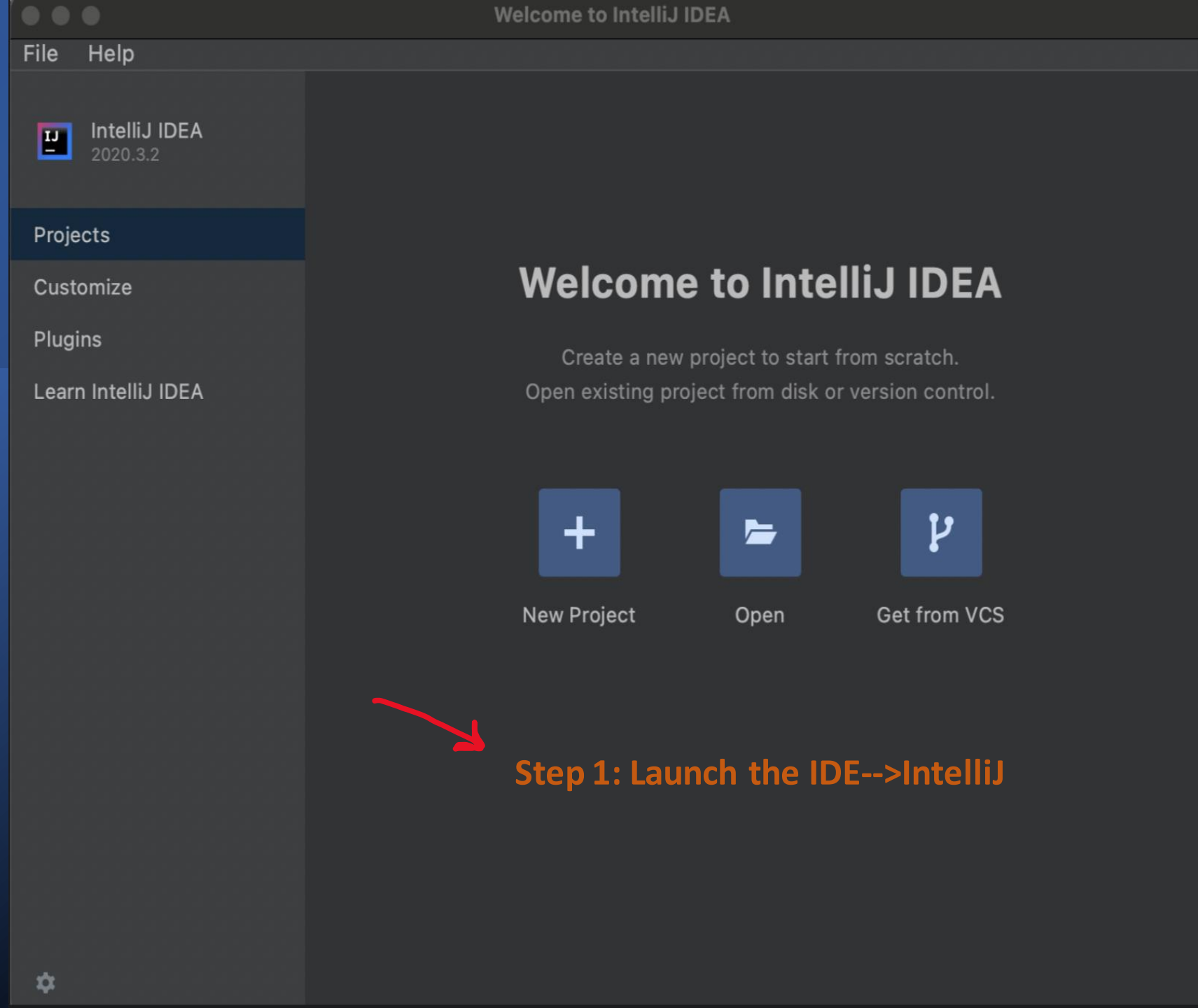
- Provides Project Structure
- Maven provides pom.xml which helps to maintain dependencies and plugin.
- Dependencies are nothing but all 3rd party jar files which helps to download jar files from the repo automatically on the other hand plugin helps to add some project related configuration. Like for compile the project we have maven compiler project, to run the entire application we have maven surefire plugin.
- Package the project
- Generate the report
- Create Project documentation also.

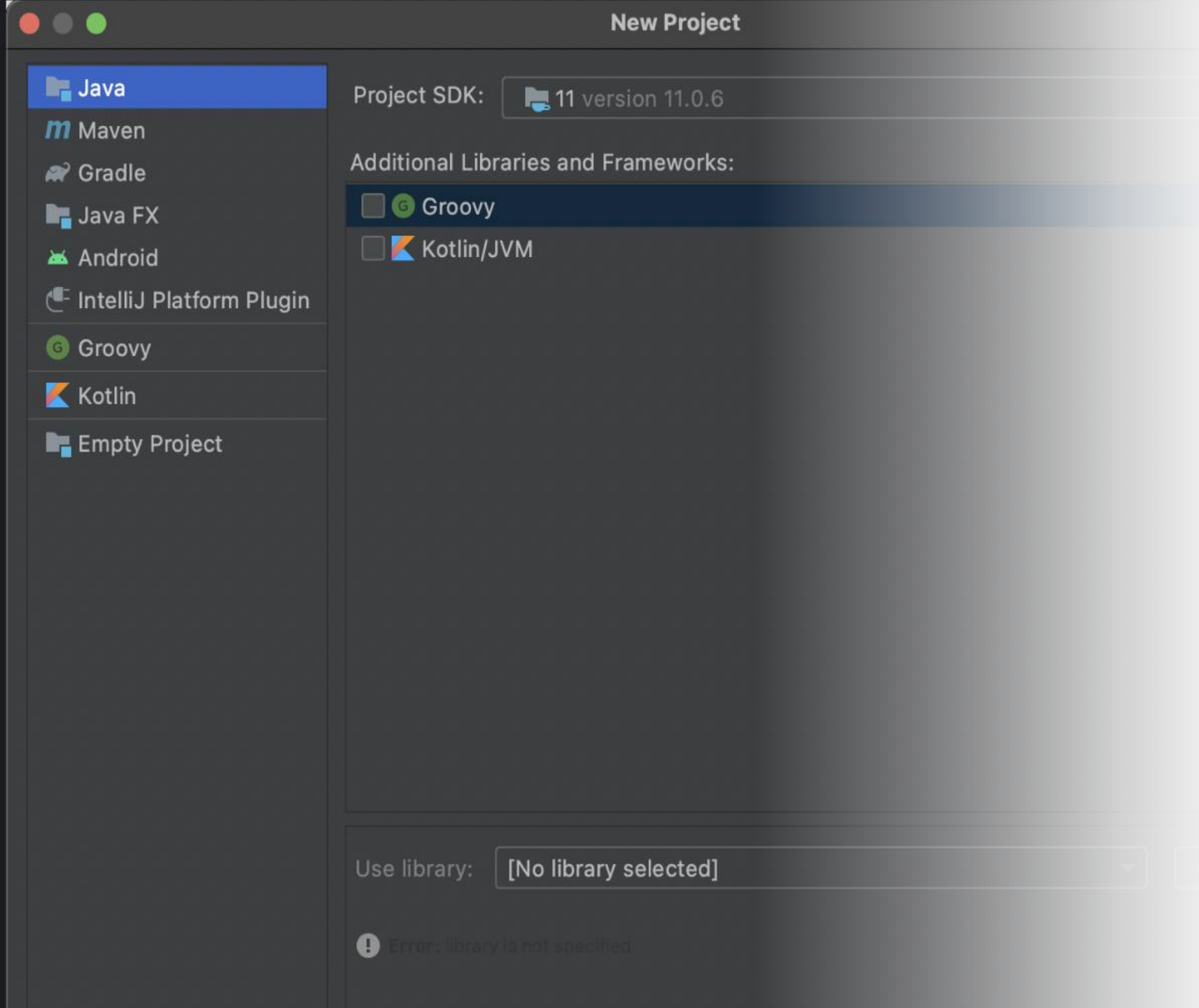


How to create Maven Project



We need to consider one IDE. I am using IntelliJ.





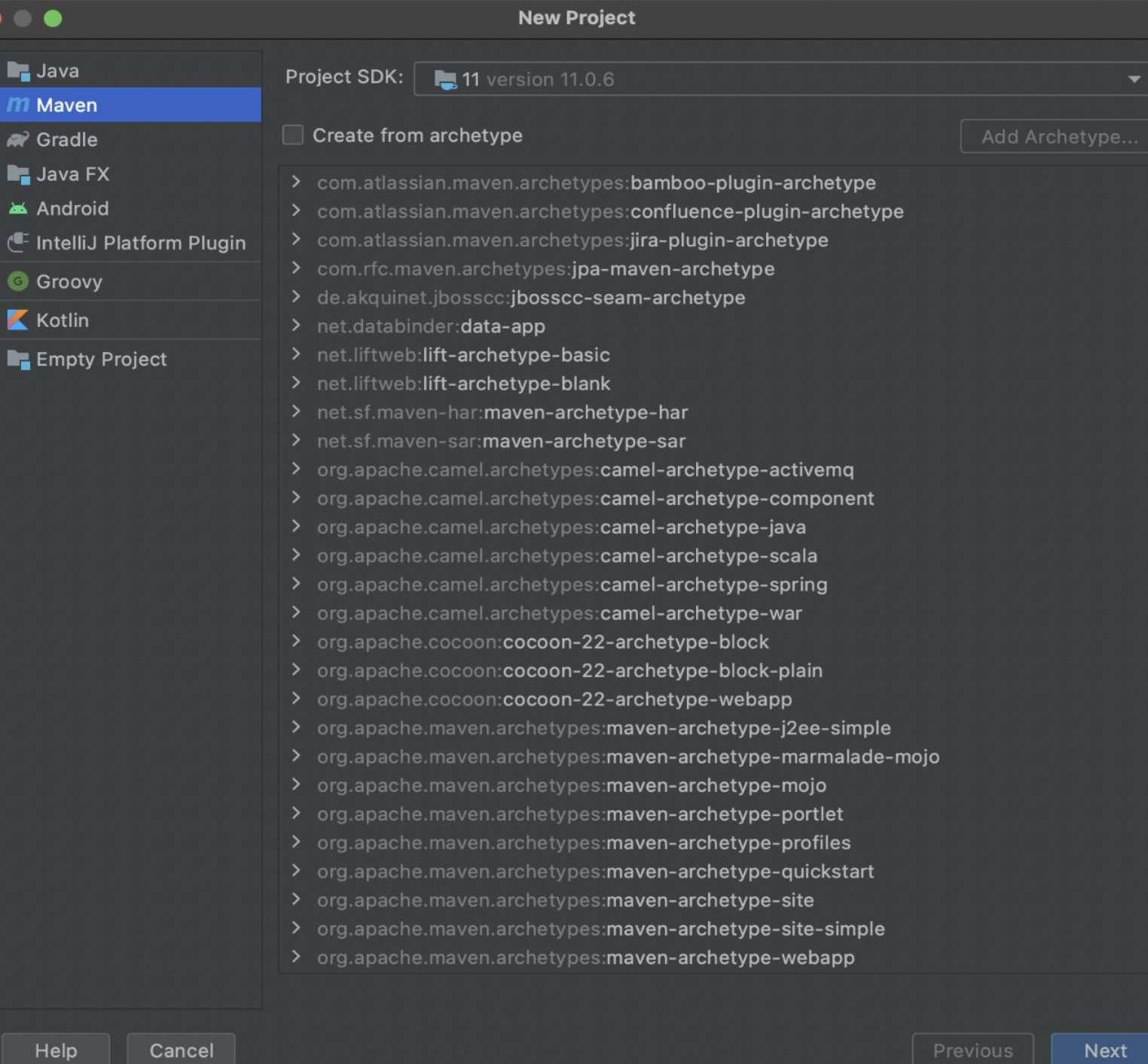
❑ Step 2:

Select Maven project and select the JDK version from the Project SDK: drop down.

❑ Note:

If we are maintaining multiple java version we can select the specific one for the project. Observe in the next slides.





❑ **Note:** Once you will select the project type maven, it will show you the list of archetypes, it is nothing but templates which is predefined and we can use as it is in our project or else we can add our own also. This defines the project structure.

❑ **Note:** if you don't want to use any existing template you can click on Next option without selecting any archetype.



New Project

Name: Demo_Test

Location: ~/Desktop/Software_Testing_Projects

▼ Artifact Coordinates

GroupId: org.TestersZone Company Name
The name of the artifact group, usually a company domain

ArtifactId: Demo_Test Project Name
The name of the artifact within the group, usually a project name

Version: 1.0-SNAPSHOT

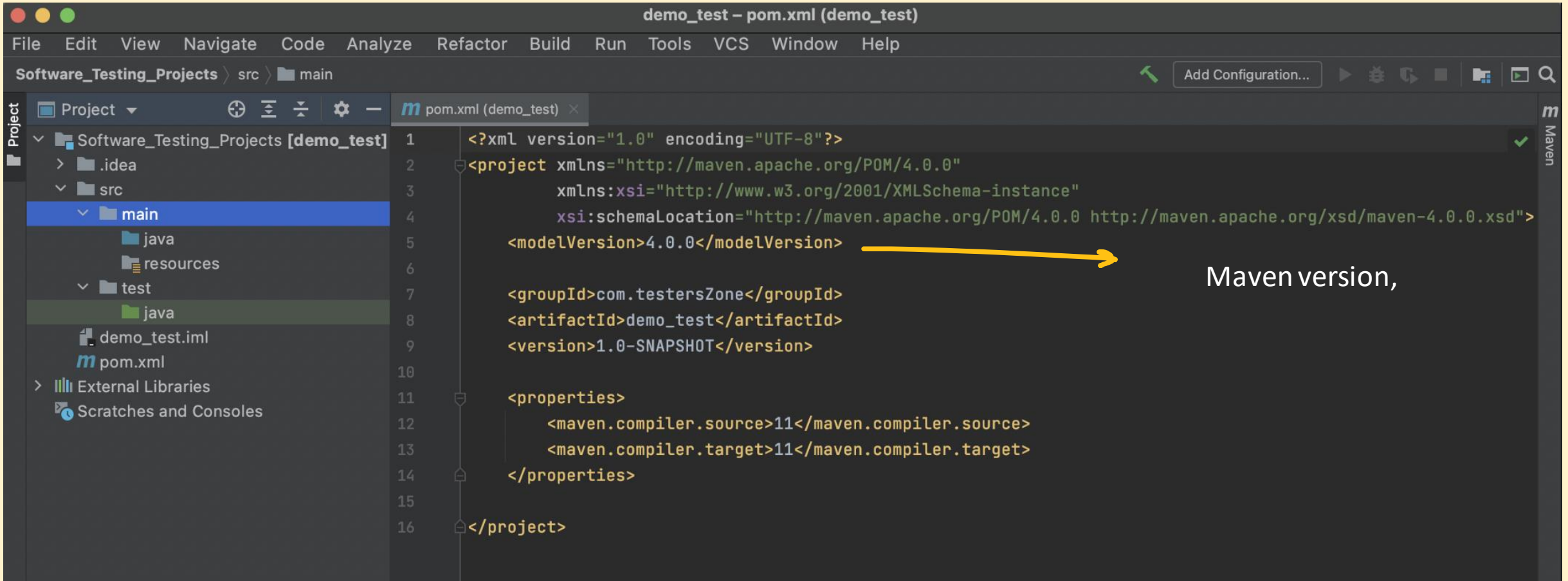
Help Cancel Previous Finish

- **Step 3:**

We have to mention GroupId, ArtifactId and Location of the project and Click finish.

GroupId --> Company Name.
ArtifactId--> Project Name.





- **Note:**

After Creating a Maven project, we will get this project structure at the end

src/main --> use to maintain the source code of the project and
src/resources use to maintain all the resources which needs for source code.
test folder use to maintain the unit code to test the developed source code.
.iml file: it is a module file, **used for keeping module configuration**. Modules allow you to combine several technologies and frameworks in one application

What is pom.xml?

pom stands for Project object model and it is one of the core part of the project.



It helps to maintain the 3rd party jar files as dependency.



It consists plugin's which needs to configure the project.



It consists meta data of the project.



Types of pom file: simple pom, super pom and effective pom.



pom file presents in project is simple pom. Super pom is parent pom and effective is combination of both.

- Where we can find Effective POM?

1. navigate to project in CLI and use this command **mvn help:effective-pom.**

2. Click on maven-->right click on project-->Show effective pom.



Self_Practice > m pom.xml

Project

- Self_Practice [Maven_Understanding] ~/Self
 - .idea
 - src
 - main
 - java
 - resources
 - test
 - java
 - Maven_Understanding.iml 15/09/21, 8:26 PM, 8
 - m pom.xml 15/09/21, 8:23 PM, 587 B 8 minutes ago
 - External Libraries
 - Scratches and Consoles

Maven_Understanding-effective-pom.xml

```
1 <!-- =====>
2 <!--
3 <!-- Generated on 20
4 <!--
5 <!-- =====
6 <!--
7 <!-- =====
8 <!--
9 <!-- Effective POM 1
10 <!--
11 <!-- 'org.TestersZon
12 <!-- =====
13
14 <project xmlns="http
15 <modelVersion>4.0.
16 <groupId>org.Teste
17 <artifactId>Maven
18 <version>1.0-SNAP
19
20 <properties>
```

Maven

- Maven_Understanding
 - Reload project
 - Generate Sources and Update Folders
 - Ignore Projects
 - Unlink Maven Projects
 - Create 'settings.xml'
 - Create 'profiles.xml'
 - Download Sources
 - Download Documentation
 - Download Sources and Documentation
 - Show Effective POM
 - Run Maven Build
 - Jump to Source

Build: Sync x

✓ Sync: At 16/09/21, 12:41 AM 1 sec, 735 ms

Structure

Favorites

TODO Problems Terminal Build

Generate and show effective POM

Event Log

1:1 LF UTF-8 4 spaces

What is

1.0-SNAPSHOT

meaning under pom.xml ?

As we know maven is a build tool, snapshot tells about the dynamic nature of the build. Since during development phase different developers will be adding the codes so jar file which will be creating at the end that is just a screenshot means not a fixed build. Final snapshot means overall development of the project and that might deploy in the production that comes under release.

So different branches with day to day development of project is just a snapshot of actual one which will be deployed after complete development.



How Maven works?



As soon we create a maven project a pom.xml file will create inside the project and also one local maven repo will create which holds all the dependencies.

All the dependencies will be getting downloaded from the Maven remote repo for first time and it will get stored in local maven repo.

Note: Internet connection is needed to download the dependencies from the maven remote repo.

How to add dependencies in the maven?

Steps:

1. Access the maven remot repo: <https://mvnrepository.com/>
2. Search the option and tap on that.
3. we will get proper maven dependency syntax which we can copy paste in our pom.xml

```
<!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.4.0</version>
  <scope>test</scope>
</dependency>
```



Note: dependencies always starts with **<dependency>** tag and ends with **</dependency>** tag. Inside that groupId, artifactId and version will be must.

Maven Dependency Scopes

Maven dependency scope attribute is used to specify the visibility of a dependency.

Maven provides six scopes

i.e. **compile, provided, runtime, test, system, and import.**

Table of Contents:

- | | | |
|-------------------|--------------------|------------------|
| 1. Compile Scope. | 2. Provided Scope. | 3. Runtime Scope |
| 4. Test Scope. | 5. System Scope | 6. Import Scope |



1. Compile Scope.



- This is **maven default scope**. Dependencies with compile scope are needed to build, test, and run the project.
- Scope compile is to be required in most of the cases to resolve the import statements into your java classes sourcecode.

```
<dependencies>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.14</version>
    <!-- You can ommit this because it is default -->
    <scope>compile</scope>
  </dependency>
</dependencies>
```



2. Provided Scope.

- **Maven dependency scope provided** is used during **build and test** the project. They are also required to run, but **should not exported**, because the dependency will be provided by the runtime, for instance, by servlet container or application server.

```
<dependency>
```

```
  <groupId>javax.servlet</groupId>
```

```
  <artifactId>servlet-api</artifactId>
```

```
  <version>3.0.1</version>
```

```
  <scope>provided</scope>
```

```
</dependency>
```

3. Runtime Scope



scope runtime are not needed to build, but are part of the classpath to **test and run** the project.

```
<dependency>
```

```
  <groupId>com.thoughtworks.xstream</groupId>
```

```
  <artifactId>xstream</artifactId>
```

```
  <version>1.4.4</version>
```

```
  <scope>runtime</scope>
```

```
</dependency>
```



4. Test Scope

scope test are not needed to build and run the project. They are needed to **compile and run the unit tests**.

```
<dependency>
```

```
  <groupId>junit</groupId>
```

```
  <artifactId>junit</artifactId>
```

```
  <version>4.12</version>
```

```
  <scope>test</scope>
```

```
</dependency>
```



5. System Scope:

dependencies with system are similar to ones with scope provided. The only difference is system dependencies are provided by us (using system path) but provided dependency always provides by jdk or server.

<dependency>

<groupId>extDependency</groupId>

<artifactId>extDependency</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>\${basedir}\war\WEB-INF\lib\extDependency.jar</systemPath>

</dependency>

The systemPath element refer to the location of the JAR file.

Types of Maven dependencies



- there're two types of dependencies in Maven direct and transitive.
- Direct dependencies are the ones that are explicitly included in the project. These can be included in the project using `<dependency>` tags:

Transitive dependencies, on the other hand, are dependencies required by our direct dependencies. Required transitive dependencies are automatically included in our project by Maven. We can list all dependencies including transitive dependencies in the project using: `mvn dependency:tree` command.



RANGE	MEANING
1.2	Version equals to 1.2 or is starting with 1.2
(,1.2]	Any version less than 1.2. Version 1.2 included. $x \leq 1.2$
(,1.2)	Any version less than 1.2. Version 1.2 excluded. $x < 1.2$
[1.2]	Only version 1.2 only. $x == 1.0$
[1.2,)	Any version greater than 1.2. Version 1.2 included. $x \geq 1.2$
(1.2,)	Any version greater than 1.2. Version 1.2 excluded. $x > 1.2$
(1.2,2.2)	Version between 1.2 and 2.2. Both excluded. $1.0 < x < 2.0$
[1.2,2.2]	Version between 1.2 and 2.2. Both included. $1.2 \leq x \leq 2.2$
(,1.2],[2.2,)	Version either less than 1.2 or greater than 2.2. Both included. $x \leq 1.2$ or $x \geq 2.2$

Maven version range

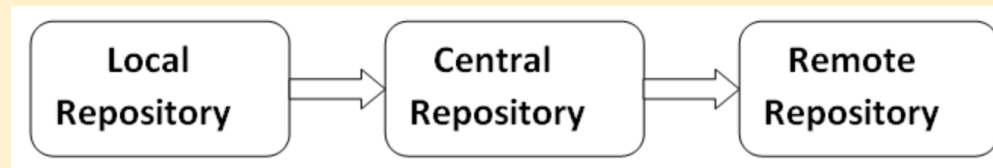


Maven Repositories

Maven repository is the place where all the packaged jar files store
There are three types of maven repositories.

1. Local Repository.
2. Central Repository.
3. Remote Repository.

dependencies Search order in the maven:





Local Repository

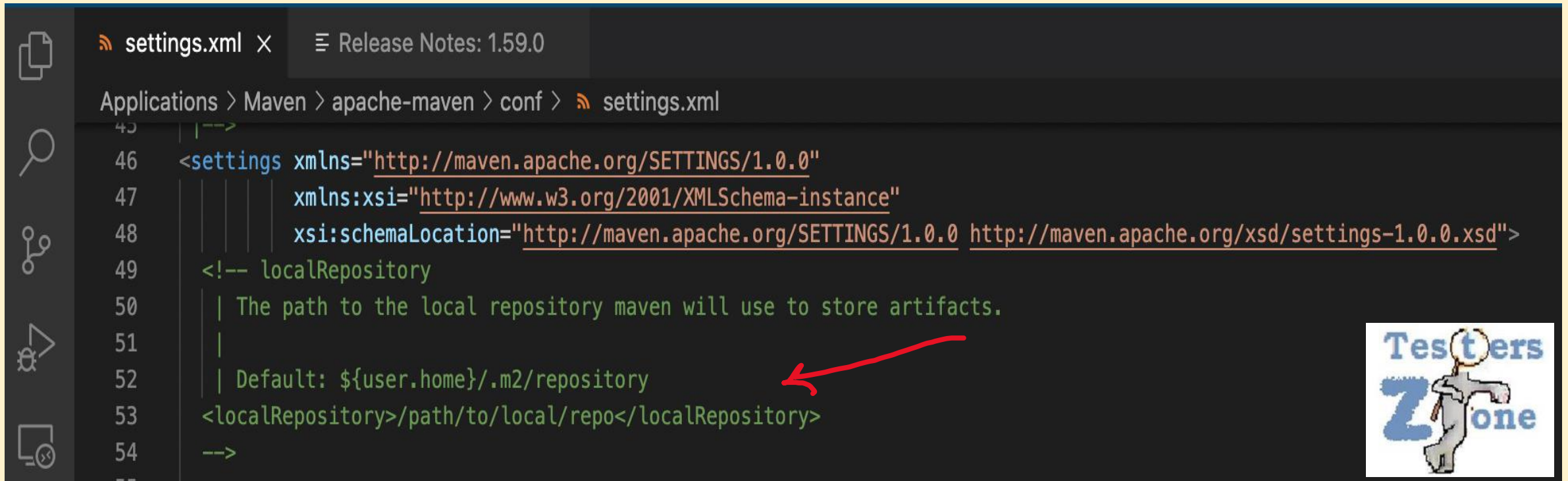
- This is created by maven in local system when any maven command runs.
- Maven repository presents under **.m2** directory [%USER_HOME%/.m2] e.g. /Users/<name_of_user>/.m2
- **Note**
Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.). When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies stored on remote machine every time a project is build.

Is it possible to "Update location of Local Repository"??


Yes, we can navigate to settings.xml file location and change the local repository path .

path: Mac--- > /Applications/Maven/apache-maven/conf/settings.xml

Path: Window--- > F:\apache-maven-3.1.1\conf\settings.xml



```
settings.xml x Release Notes: 1.59.0
Applications > Maven > apache-maven > conf > settings.xml
45 | -->
46 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
47 | | | | xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
48 | | | | xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
49 | | | | <!-- localRepository
50 | | | | | The path to the local repository maven will use to store artifacts.
51 | | | | |
52 | | | | | Default: ${user.home}/.m2/repository
53 | | | | <localRepository>/path/to/local/repo</localRepository>
54 | | | | -->
```



Central Repository



- Maven **central repository** is located on the web. It has been created by the apache maven community itself.
- The path of central repository is: <http://repo1.maven.org/maven2/>.
- Key concepts of Central repository are as follows:
 - It is not required to be configured.
 - It requires internet access to be searched.
 - To browse the content of central maven repository, maven community has provided a URL – <https://search.maven.org/#browse>. Using this library, a developer can search all the available libraries in central repository.

Remote Repository



- Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of **Remote Repository**, which is developer's own custom repository containing required libraries or other project jars.
- You can configure a remote repository in the POM file or super POM file in remote repository itself.

```
<repositories>  
  <repository>  
    <id>org.source.repo</id>  
    <url>http://maven.orgName.com/maven2/</url>  
  </repository>  
</repositories>
```


Maven Dependency Search Sequence



When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence –

- **Step 1** – Search dependency in local repository, if not found, move to step 2 else perform the further processing.
- **Step 2** – Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4. Else it is downloaded to local repository for future reference.
- **Step 3** – If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
- **Step 4** – Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

Notes:



- A **namespace** is used to uniquely identify one or more names from other similar names of different objects.
- In xml all the elements which we use those are defined by w3c in namespace i.e. <http://www.w3.org/2001/XMLSchema>
we can use all the elements with fully qualified name. Fully qualified means elements name along with namespace.
- We can also import this namespace into xmlns and use this as a short name to avoid long fully qualified names.
- SchemaLocation helps to map the xml to the xsd(xml schema directory).
E.g. <schemaLocation="target namespace of xsd, xsd filename">
- SchemaLocation attribute is given by w3c in namespace <http://www.w3.org/2001/XMLSchema-instance>

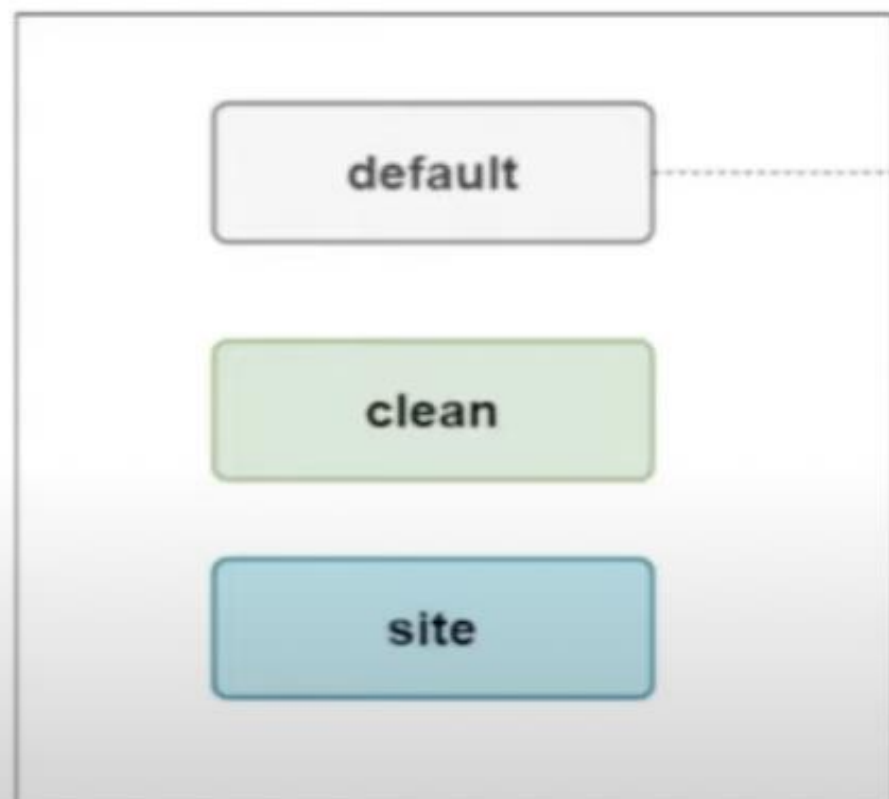
Maven Build Life Cycle



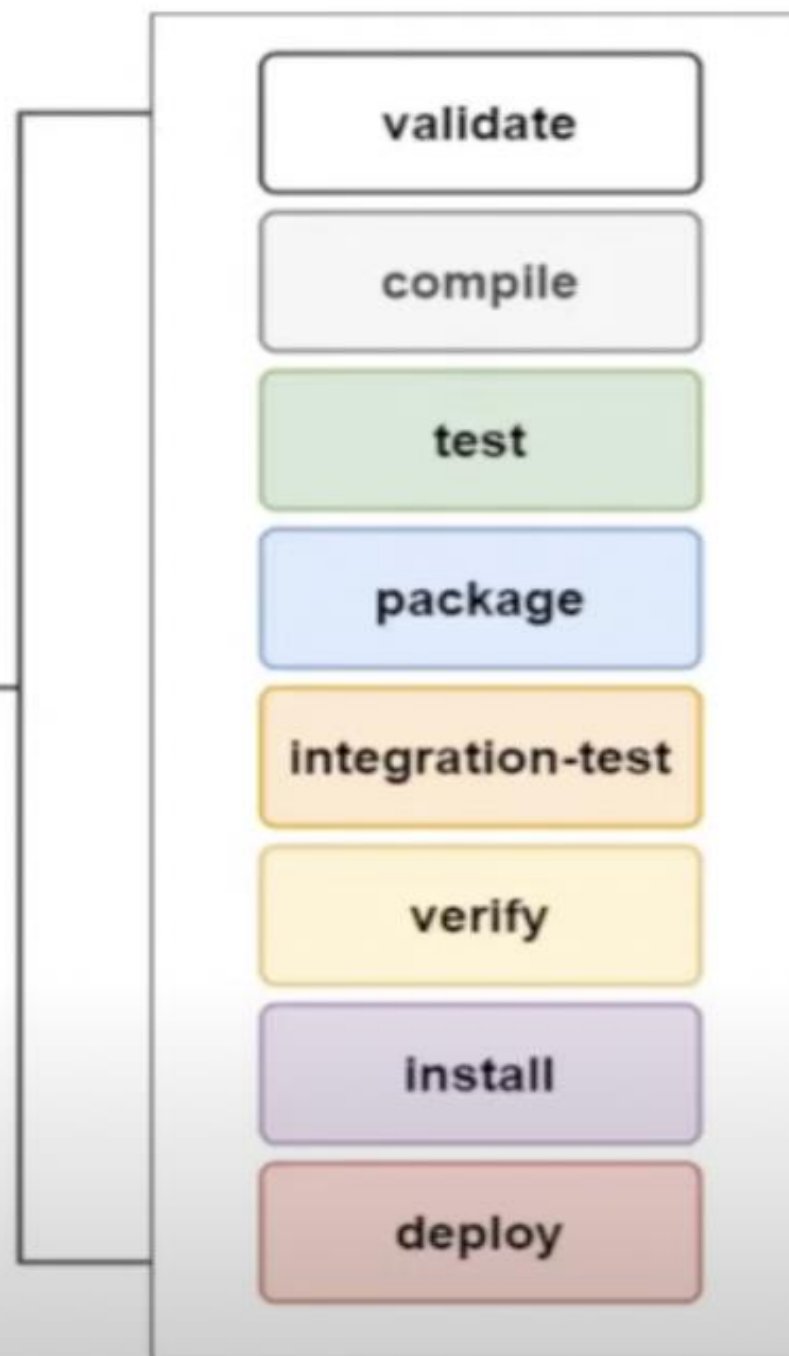
Maven Build Lifecycle, Phase, Goal

- Maven build lifecycle is the set of tasks that are executed when the maven build is run.
- There are three built-in build lifecycles:
 - default
 - clean
 - site
- Maven build lifecycle goes through a set of stages, they are called build phases.
- A build phase is made up of a set of goals. Maven goals represent a specific task that contributes to the building and managing of a project.

Build Lifecycle



Lifecycle Phases



validate – verifies whether the pom.xml file is valid or not

compile – compiles the source code inside the project

test – runs unit-tests inside the project

package – packages the source code into an artifact (ZIP, JAR, WAR or EAR)

integration-test– process and deploy the package if necessary into an environment where integration tests can be run.

verify – checks whether the created package is valid or not.

install – installs the created package into our **Local Repository**

deploy – deploys the created package to the **Remote Repository**



Note



Each Lifecycle is independent of each other and they can be executed together.

The **clean** lifecycle is mainly responsible to clean the **.class** and meta-data generated during default life cycle phases.

The **site** lifecycle phase is responsible to generate Java Documentation.

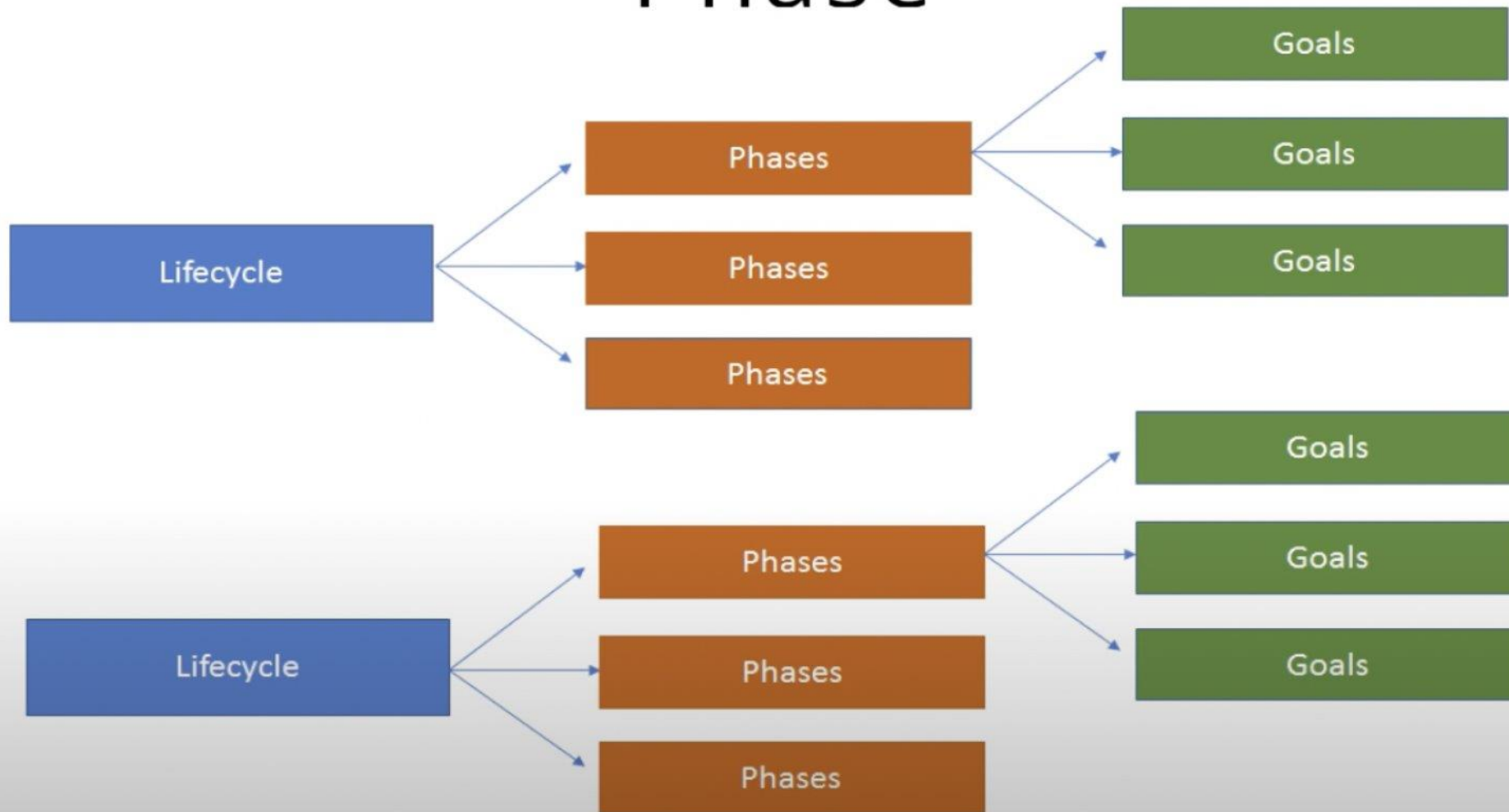
To be able to execute these Lifecycle Phases, Maven provides us with **Plugins** to perform each task.

Each plugin is associated with a particular **Goal**

Plugins and Goals



Phase



- **Goal** defines the task which needs to be performed once we execute the maven command,
- **Each phase is a sequence of goals, and each goal is responsible for a specific task.**
- When we run a phase - all goals bound to this phase are executed in order.
- **Compiler:compile** - the compile goal from the compiler plugin is bound to the compile phase
- **compiler:testCompile** - is bound to the test-compile phase
- **Surefire:test** - is bound to test phase
- **Install:install** - is bound to install phase
- **Jar:jar** and **war:war**. - is bound to package phase



**** We can list all goals bound to a specific phase and their plugins using the command:**
mvn help:describe -Dcmd=PHASENAME.

e.g. **mvn help:describe -Dcmd=compile**

Output:

compile' is a phase corresponding to this plugin:

org.apache.maven.plugins:maven-compiler-plugin:3.1:compile

Which, as mentioned above, means the compile goal from compiler plugin is bound to the compile phase.

**** We can use the following command to list all goals in a specific plugin:**

mvn <PLUGIN>:help

e.g. **mvn surefire: help**

**** We can also run only a specific goal of the plugin:**

e.g. **mvn compiler:compile** ← goal

↑
Plugin



Maven Compile Plugin

- The Maven Compile Plugin is responsible to compile our Java files into the .class files.
- You can add this plugin to your project by adding the below section to your **pom.xml** file under the **dependencies** section

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
    </plugin>
  </plugins>
</build>
```



Maven Surefire Plugin

- Using **Surefire Plugin**, we can run the tests inside our project by using the following command:
\$ mvn test

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M5</version>
    <configuration>
      <suiteXmlFiles>
        <suiteXmlFile>testerszone.xml</suiteXmlFile>
      </suiteXmlFiles>
    </configuration>
  </plugin>
</plugins>
```



Maven Install Plugin

This is used to package our source code into an artifact of our choice like a JAR and install it to our **Local Repository** which is `<USER_HOME>/.m2/repository` folder.

```
$ mvn install
```

```
<plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
```

```
<artifactId>maven-install-plugin</artifactId>
```

```
<version>2.5.2</version>
```

```
</plugin>
```



Note:

you can observe that the **install** phase includes also the previous phases in the lifecycle, so as part of this phase maven:

validates our pom.xml (**validate**)

compiles our source code (**compile**)

executes our tests (**test**)

packages our source code into JAR (**package**)

installs the JAR into our local repository (**install**)



Maven Deploy Plugin



The Maven Deploy Plugin will deploy the artifact into a remote repository.

Note: The deploy goal of the plugin is associated with the **deploy** phase of the build lifecycle.

Before running the **deploy** phase of the lifecycle, we have to make sure that the remote repository details are configured inside our project.

```
<plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
```

```
<artifactId>maven-deploy-plugin</artifactId>
```

```
<version>2.8.2</version>
```

```
</plugin>
```

Note:

Before running the **deploy** phase of the lifecycle, we have to make sure that the remote repository details are configured inside our project.

Note:

We can configure this details inside the **distributionManagement** section:

```
<distributionManagement>
  <repository>
    <id>test-distribution</id>
    <name>Testers Zone</name>
    <url>http://testers-zone</url>
  </repository>
</distributionManagement>
```



Note: To be able to connect to the remote repository, maven needs access to the credentials, which can be configured inside a special file called as **settings.xml** file.

```
<settings
  xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>test-distribution</id>
      <username>my_username</username>
      <password>my_password</password>
    </server>
  </servers>
</settings>
```

How to execute testng suite using maven?



1. Add this snippet of code under pom.xml

```
<plugins>
  <!-- Following plugin executes the testng tests -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.14.1</version>
    <configuration>
      <!-- Suite testng xml file to consider for test execution -->
      <suiteXmlFiles>
        <suiteXmlFile>testng.xml</suiteXmlFile>
        <suiteXmlFile>suites-test-testng.xml</suiteXmlFile>
      </suiteXmlFiles>
    </configuration>
  </plugin>
</plugins>
```

2. use maven command **mvn test** to execute the test cases mentioned in the testng xml file

Pom.xml default code explanation

```
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

******Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax: **xmlns="namespaceURI"**

In the attribute **xmlns:pxf**, xmlns is like a reserved word, which is used only to declare a namespace. In other words, xmlns is used for binding namespaces, and is not itself bound to any namespace. Therefore, the above example is read as binding the prefix "pxf" with the namespace "<http://www.foo.com>."

In Java : String pfx = "<http://www.library.com>"

In XML : <someElement xmlns:pxf="http://www.foo.com" />

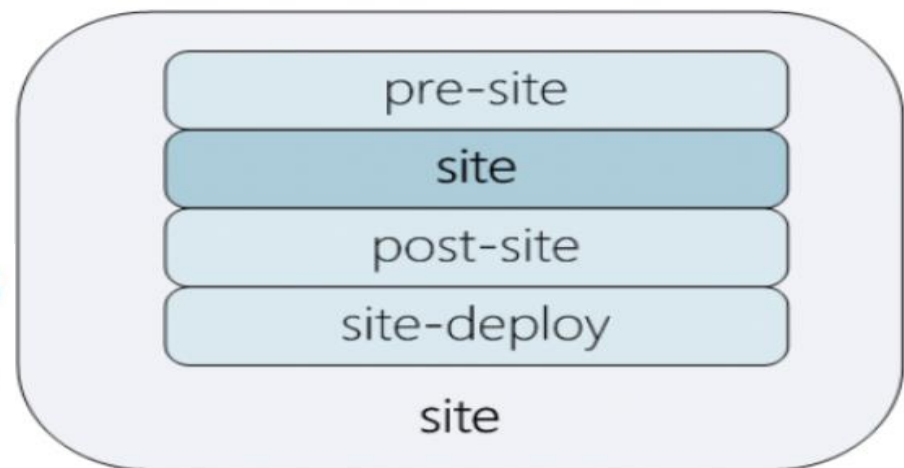
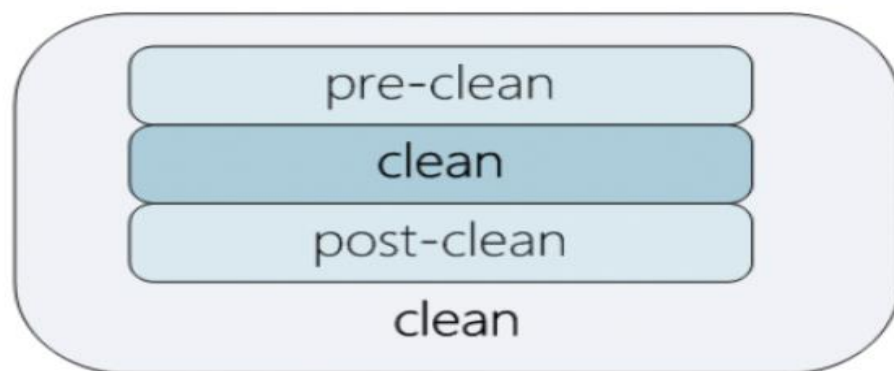
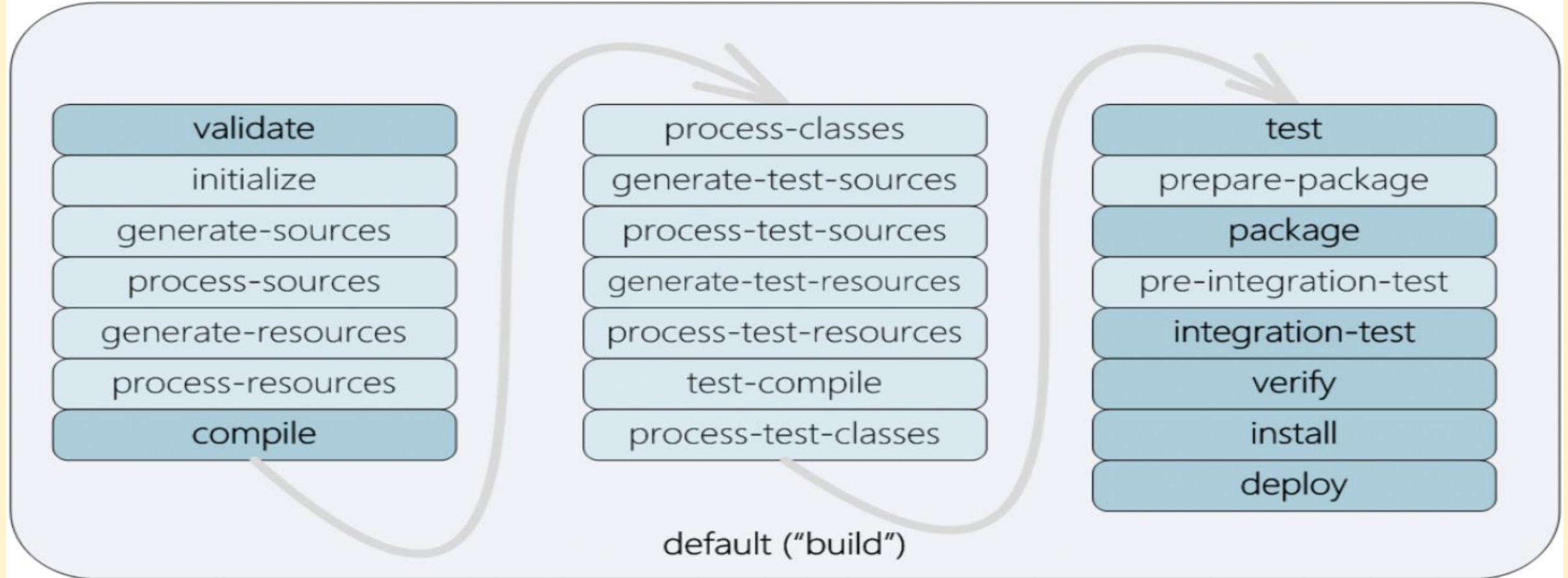
****** A package in Java can have many reusable classes and interfaces. Similarly, a namespace in XML can have many reusable elements and attributes.

To get the more details on xmlns, xsd or schemaLocation you can access this link: <https://www.tutorialspoint.com/xsd/index.htm>

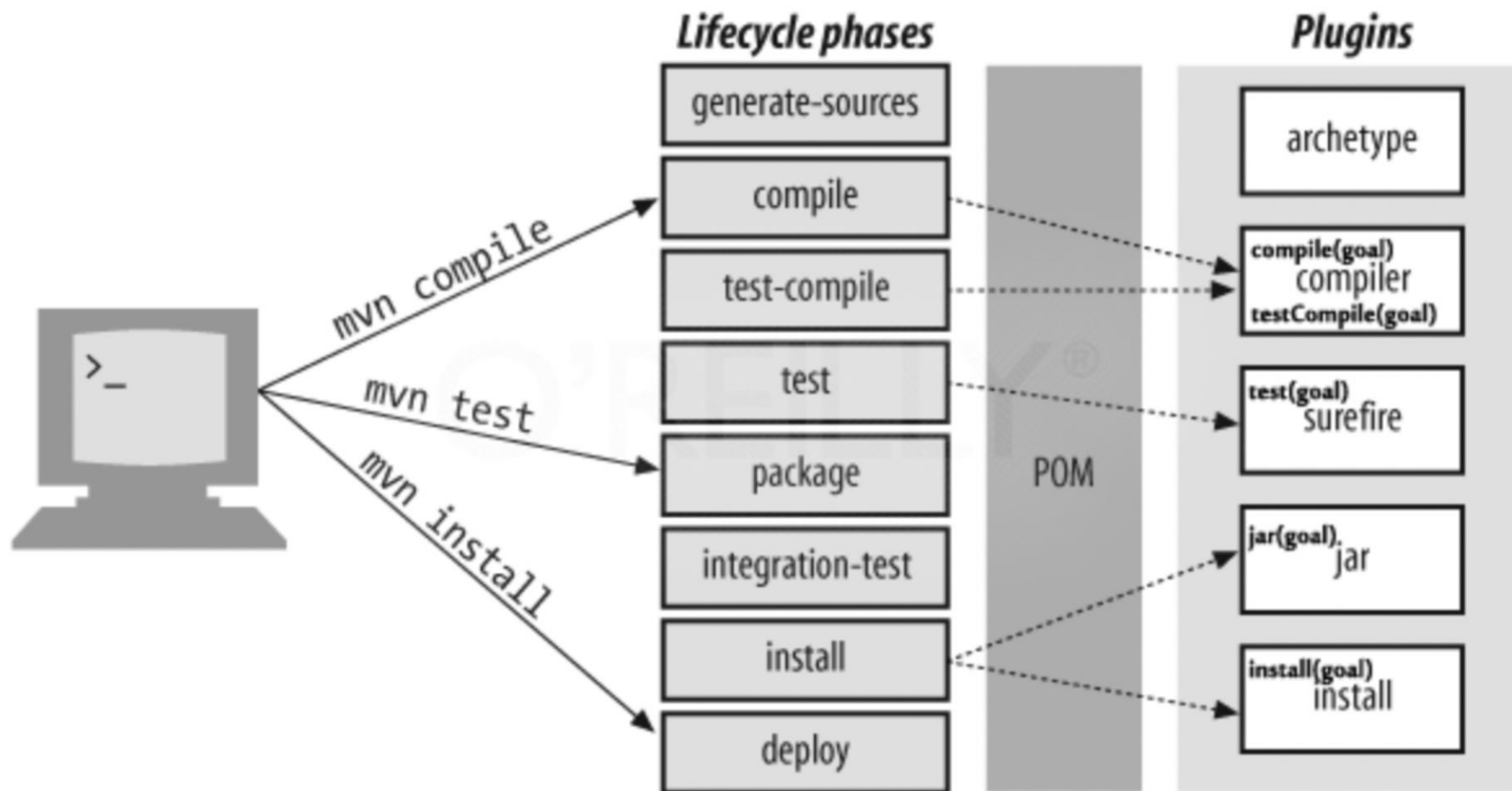


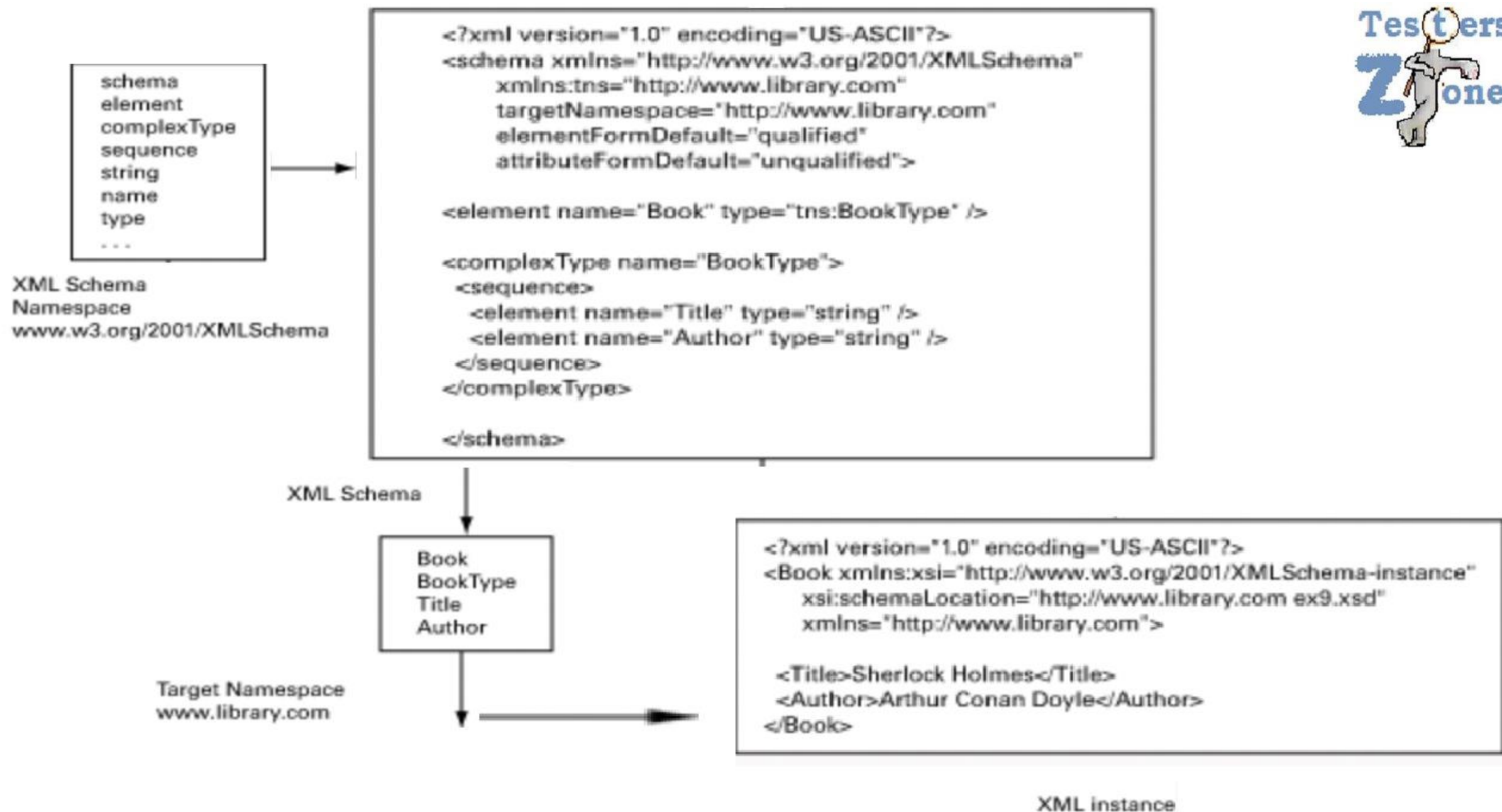
-
- +
 - Few attachments for the learning reference





Therefore, **phase** is made up of **plugin goals**.





Thank you



Mithilesh Singh