



**MAVEN**

Avinash Patel  
Senior Manager



# Agenda

1 Maven Introduction

2 Maven POM File & Project Structure

3 Create, Build & Test Maven Project

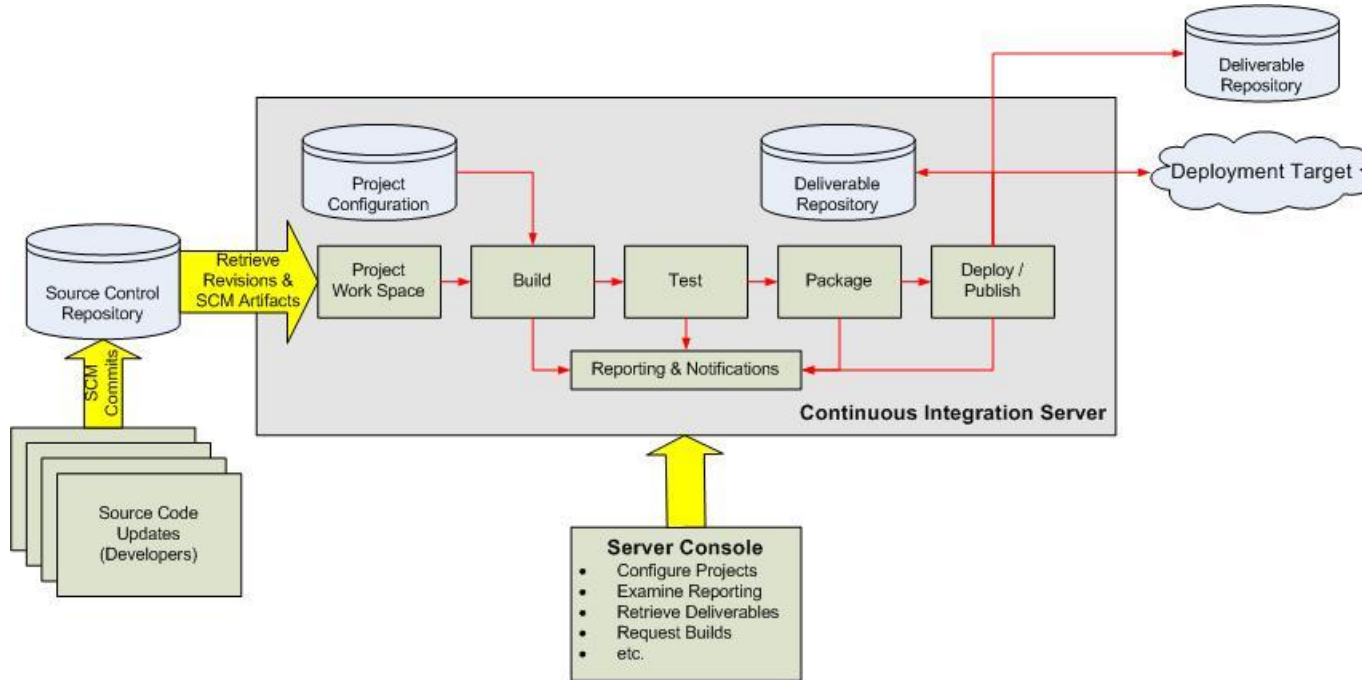
4 Maven Build Life Cycle

5 Lab/Exercise

# Objectives

- Understand the challenges of Project Development Build
- Recognize the need of a tool to automate build process & Managing projects
- Configure Maven and check the set up
- Describe the features of Maven

# Continuous Integration Overview



source: <http://www.javaworld.com/javaworld/jw-12-2008/images/CIOverview.jpg>

# Maven Introduction

Overview & Features



# Challenges while Building a Project

- **Lack of Uniform directory Structure:** The team needs a uniform directory structure of the Project while working in a project
- **Too many dependencies on Jar file:** During integration injecting proper dependency and jar files would always be a challenge as it is difficult to assess transitive dependency at first place.
- **Building and deploying Project:** Even though build tools like ant can solve this problem in a limited way, a proper life cycle management of Build has been the need of the hour.

# Maven Overview

- Maven is a Java build tool
  - It is a project management & comprehension tool
  - It used for building & managing any Java-based project.
  - It stores libraries & plugins in a Maven central repository
  - Maven Home Page URL: <https://maven.apache.org>. One can find the reference documentation for Maven & the core Plugins
- Why Maven?
  - Fairly large software projects generally contain many projects/modules
  - Build process without automation in such projects, is challenging to understand and maintain it with in stipulated time.

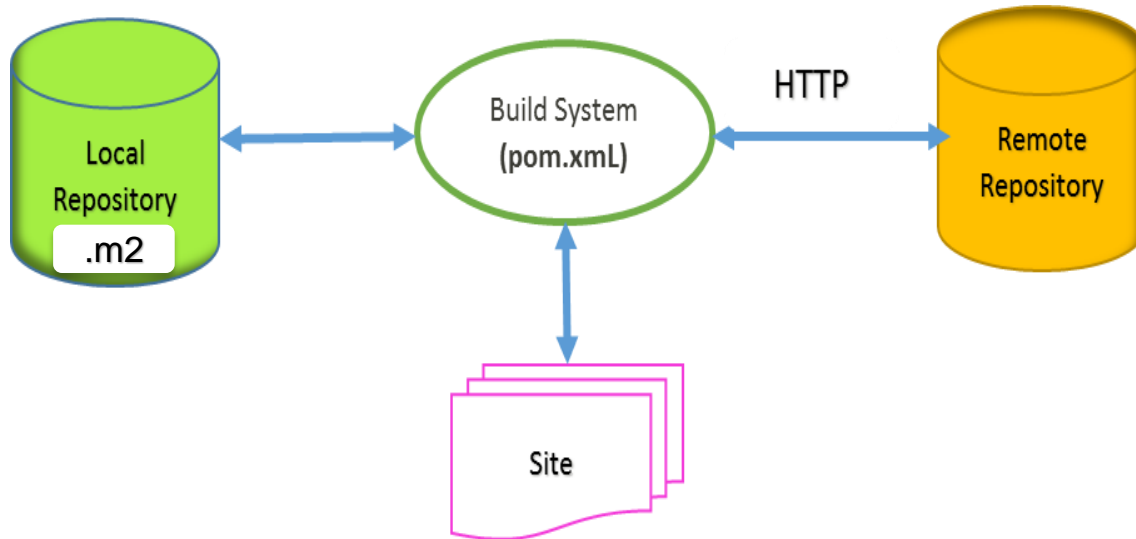
# Features of Maven

- Easier configuration
  - Consistent project structure
  - Provides project templates known as archetypes.
- Dependency Management
  - Multi-module builds
  - Build model can be sliced and diced
  - During Maven build process the dependencies are resolved
- Repository
  - Project dependencies can be accessed from local file system and/or from central or from public repositories.
- Plugin oriented - Extensible via plug-ins
  - This feature helps to keep Maven core small with basic functionalities
    - E.g. Maven core doesn't know how to compile the available Java source code. Compilation is handled by compiler plug-in
  - Additional features can be easily linked with suitable plug -ins
- Maven Central
  - Maven central repository is an open repository.
  - Maven's central repository hosts libraries. These libraries can be used in your build.
  - Maven build uses by default the central library to search for required libraries.



# Build Process

- Maven **P**roject **O**bject **M**odel (POM) is a fundamental unit of work in Maven
  - POM is an XML file.
  - This XML file contains information about project & configuration details



# Lab 1: Environment Setup



# Objectives

- Install and set environment variables for using Java & Maven

# Step 1: Java Configuration

- System Requirement
  - JDK version: 1.5 or above
  - Memory / Disk Space : no minimum requirement.
- Set JAVA Environment
  - Set environment variable
    - **JAVA\_HOME** – This variable points to the base directory location  
E.g. JAVA\_HOME = C:\Program Files\Java\jdk1.x.x
    - Append location of Java compiler System variable 'Path'  
E.g. Path = %path%;C:\Program Files (x86)\Java\jdk1.8\bin
- Verify Java & your settings on your machine
  - C:\Users\avitepa>java -version  
java version "1.8.0\_121"  
Java(TM) SE Runtime Environment (build 1.8.0\_121-b13)  
Java HotSpot(TM) Client VM (build 25.121-b13, mixed mode)
  - C:\Users\avitepa>path
  - C:\Users\avitepa>echo %classpath%

## Step 2: Maven Setup

- Download Maven - <http://maven.apache.org/download.cgi>
  - Windows: apache-maven-<<Version>>-bin.zip
- Extract/ Unzip => apache-maven-<<Version>>--bin.zip
  - C:\avitepa\apache-maven-<<Version>>
- Include bin directory of the just created directory to 'PATH' environment variable
- Verify Maven installation => C:\Users\avitepa>mvn -v

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5;2015-11-10T22:11:47 + 05:30)
Maven home: C:\apache-maven-3.3.9
Java version:1.8.0_121, vendor: Oracle Corporation
Java home:C:\Program Files (x86)\Java\jdk 1.8.0_121\jre
Default locale:en_US, platform encoding: Cp1252
OS name:"windows 8", version: "6.2", arch: "x86", family: "windows"
```

## Step 3: Maven Configuration

- set M2\_HOME=C:\avitepa\apache-maven-<<Version>>
  - M2 is the variable defined while setting Maven environment variables
  - Append string %M2% to end of system variable 'PATH'
- verify settings with 'set M' command to list environment variables starting with 'M'

```
C:\Users\avitepa>set M
```

```
M2_HOME=C:\apache-maven-3.3.9
```

```
MAVEN_HOME=C:\apache-maven-3.3.9
```

```
MAVEN_OPTS=Xms256m -Xmx1G -XX:PermSize = 512m -noverify
```

Note: If you get "out of memory" errors when running your projects, the specified memory settings can be changed as required.

# Understanding Environment Variables

- JAVA\_HOME stores the path of the directory in which JDK can be found.
- M2\_HOME stores the path of top directory in which MAVEN is "installed" (or unzipped).
- The M2 directory indicates to maven application (mvn) regarding where the required maven repositories are found
- MAVEN\_OPTS - Specify Java command line arguments which will be in effect for the execution of Maven itself
  - -Xms – initial heap size
  - -Xmx - maximum heap size

# Quiz

1. Command to check if Maven is properly installed is

- a) java -version
- b) mvn -v
- c) mvn - version

2. Which of the following environment variable points to top directory in which maven is installed:

- a) JAVA\_HOME
- b) MAVEN\_HOME
- c) M2\_HOME



# Quiz

3) POM is a

- a) XML file
- b) Word doc
- c) Excel file

# POM



# Objectives

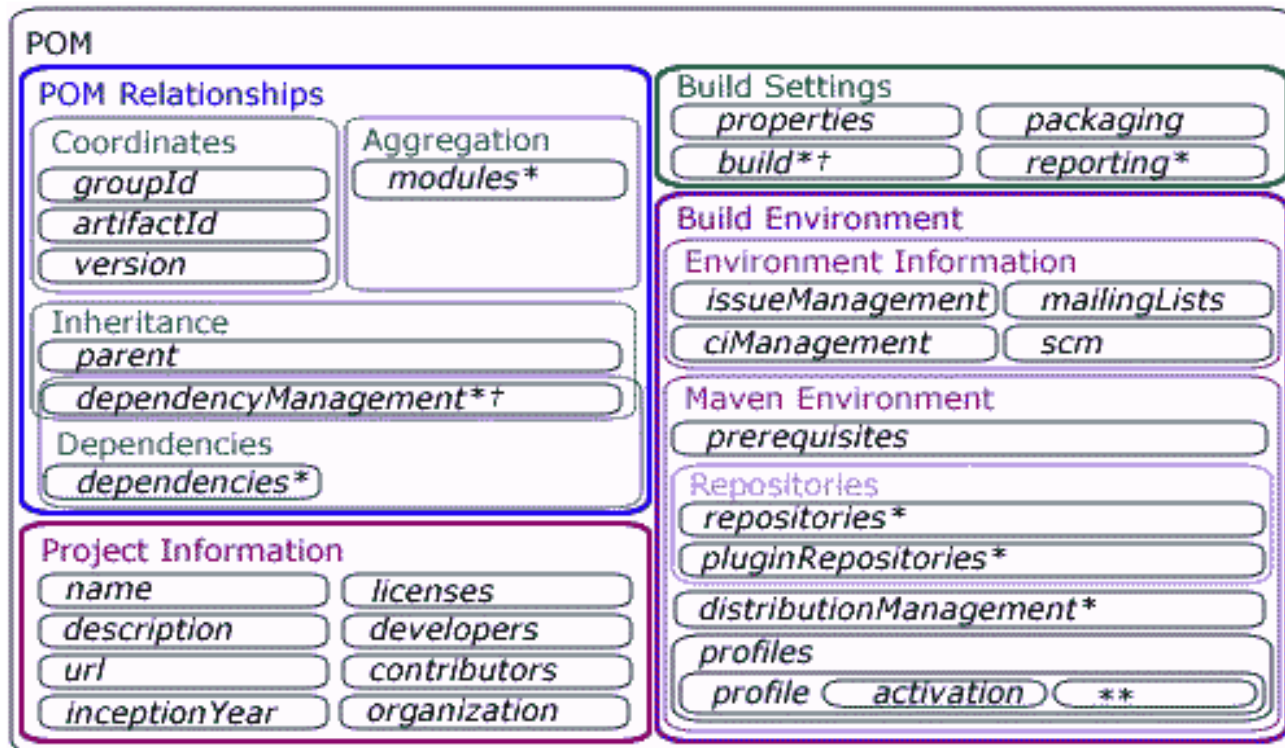
At the end of this module you would be able to

- Recognize POM structure
- Describe mandatory fields of POM
- Create a Project Structure
- Understand the hierarchy of the folders

# Maven POM

- **P**roject **O**bject **M**odel (POM)
  - pom.xml describes a project in terms of its,
    - Name and Version
    - Artifact Type
    - Source Code Locations
    - Dependencies
    - Plugins
    - Commands (goals) that can be executed
    - Profiles (Alternate build configurations)
- POM uses standard build order, directories and plugins
- Pom.xml identifies the dependencies provides uniformity and enables the build process easier
- A POM can call it's child POMs

# POM - Structure



\* Element may be overridden (at least mostly) by *profile* element settings

\*\* Profile elements are the \*-suffixed elements

† Contains elements for meant for inheritance

# Mandatory Fields -POM

- groupId

- A Maven project has a group ID.
- This ID is Unique amongst Organization
- It represents the id for group project.

- artifactId

- artifactId is the id of the project i.e. the name of Project

- Version

- It gives information whether it is a development version or release version etc. For example: 1.0-SNAPSHOT signifies it is development version.

# POM - Example

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.avitepa.FirstProject</groupId>
  <artifactId>FirstProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>FirstProject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

groupId: Id of Project group

artifactId: Id of project

version: version of project

**Note: Project Notation in repository -**  
groupId:artifactId:version

# Lab 2: Create & Build Maven Project





# Objectives

- Build a java project by creating a simple Maven project
- Perform Compile, unit test & package the result

# Maven Project Structure Creation

- Maven provides "standardized" folder structure for different kinds of software projects.
- Create a maven project by using archetype plugin and by specifying suitable goal
  - Maven generation process switches into interactive mode to make relevant settings If goals are not specified,
  - Maven provides archetypes from a Java app to a complex web app

E.g.: `mvn archetype:generate`

- `maven-archetype-quickstart`: It generates a sample Maven project.
- `maven-archetype-simple`: It generates a simple Maven project.
- `maven-archetype-webapp`: It generates a sample Maven Webapp project.

For more information on archetype, visit

<https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>

# Maven Project Creation & Execution

## 1. Create Maven Project

- a) `mvn archetype:generate -DgroupId=com.avitepa.FirstProject -DartifactId=FirstProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
- b) Go through the generated project
- c) Go through the generated POM

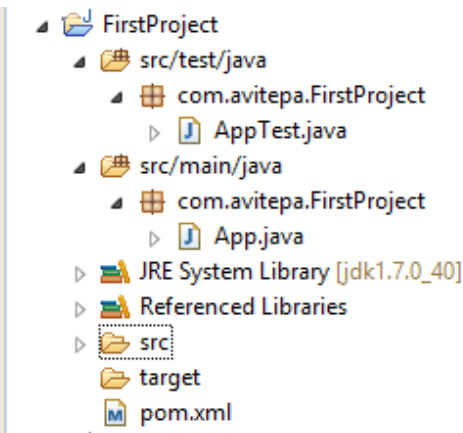
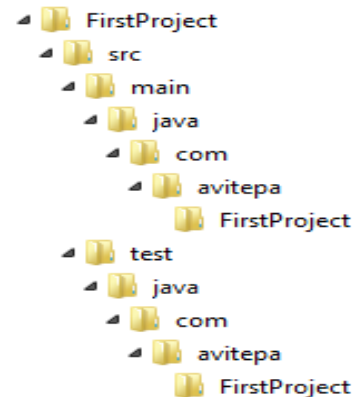
## 2. Make Maven Project as an Eclipse project

Go to the Project folder created in the above step & then run - `mvn eclipse:eclipse`

site/clean, clean package

# Maven Project Creation & Execution contd..

- Compile your sources
  - mvn compile
- Create a JAR file
  - mvn package
- Run the program
  - `..\FirstProject\target\classes>java com.avitepa.FirstProject.App`  
Hello World!
- Clean the project by removing all build
  - mvn clean
  - Note: Invokes just clean
- Rebuild
  - mvn package
  - Mvn clean package
- Running the test
  - mvn test
- Create a report
  - Mvn site



# Review generated POM

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.avitepa.FirstProject</groupId>
  <artifactId>FirstProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>FirstProject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

groupId: Id of Project group

artifactId: Id of project

version: version of project

Note: Project Notation in repository -  
groupId:artifactId:version

# Update POM

- Generated pom.xml, may have mappings to JDK 1.4 & Junit test for version 3.8.x
- Let us update the versions of JDK and Junit.



pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.7</source>
        <target>1.7</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

JDK configured to 1.7

## Update POM (contd..)

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```



Version changed to 4.11

# Compile, Run & Test

- Compile your sources
  - mvn compile
- Create a JAR file
  - mvn package
- Run the program
  - `..\FirstProject\target\classes>java com.avitepa.FirstProject.App`  
Hello World!
- Clean the project (Remove entire build results)
  - mvn clean
  - Note: Invokes just clean
- Rebuild
  - mvn package
  - Mvn clean package
- Running the test
  - mvn test
- Create a report
  - Mvn site



# Quiz

1) .m2 repository which is local repository can be found inside

- a) Maven home directory
- b) Maven bin folder
- c) users home directory
- d) None of the above

2) A unit of work/ task in Maven is called as

- a) Task
- b) goal

# Quiz

3) Give the proper sequence of attributes of POM file which identifies an Object properly

- a) Version
- b) groupId
- c) artifactId

# Maven Lifecycle



# Objectives

At the end of this section you would be able to

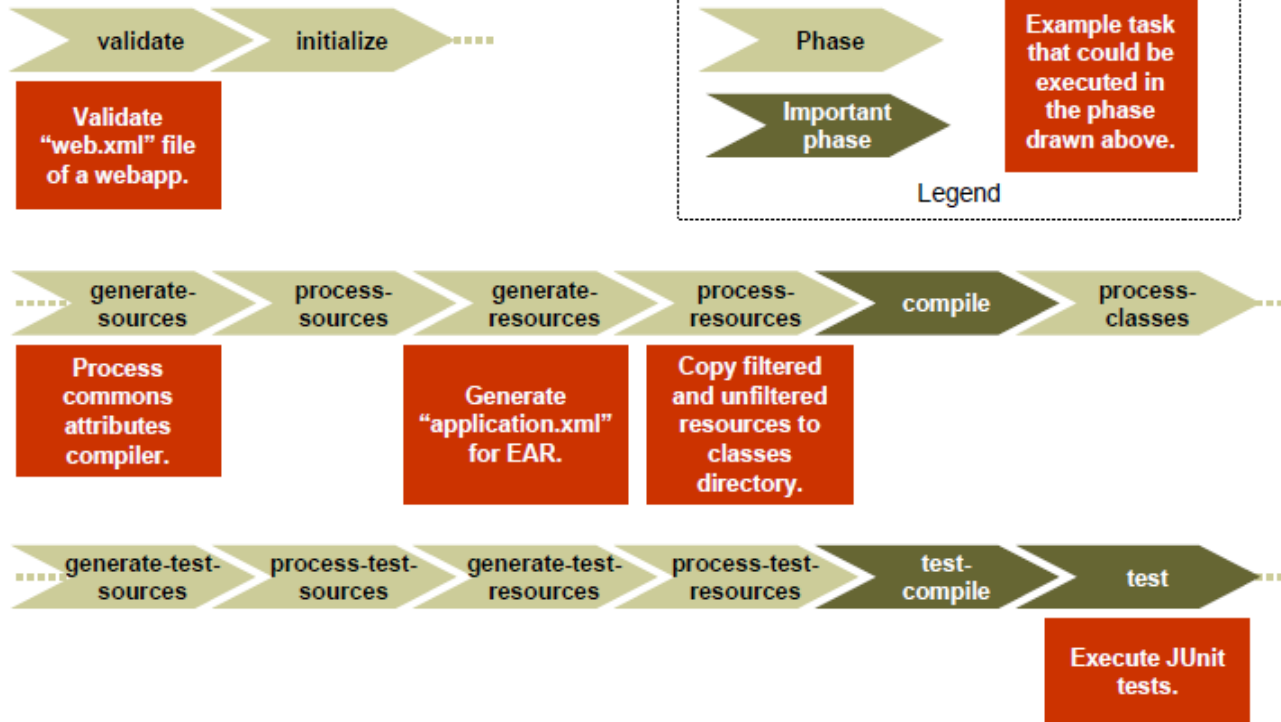
- Comprehend different phases of Maven build life cycle
- Have an understanding of the commands associated with the build
- Explain command hierarchy.

# Maven plug-ins & goals

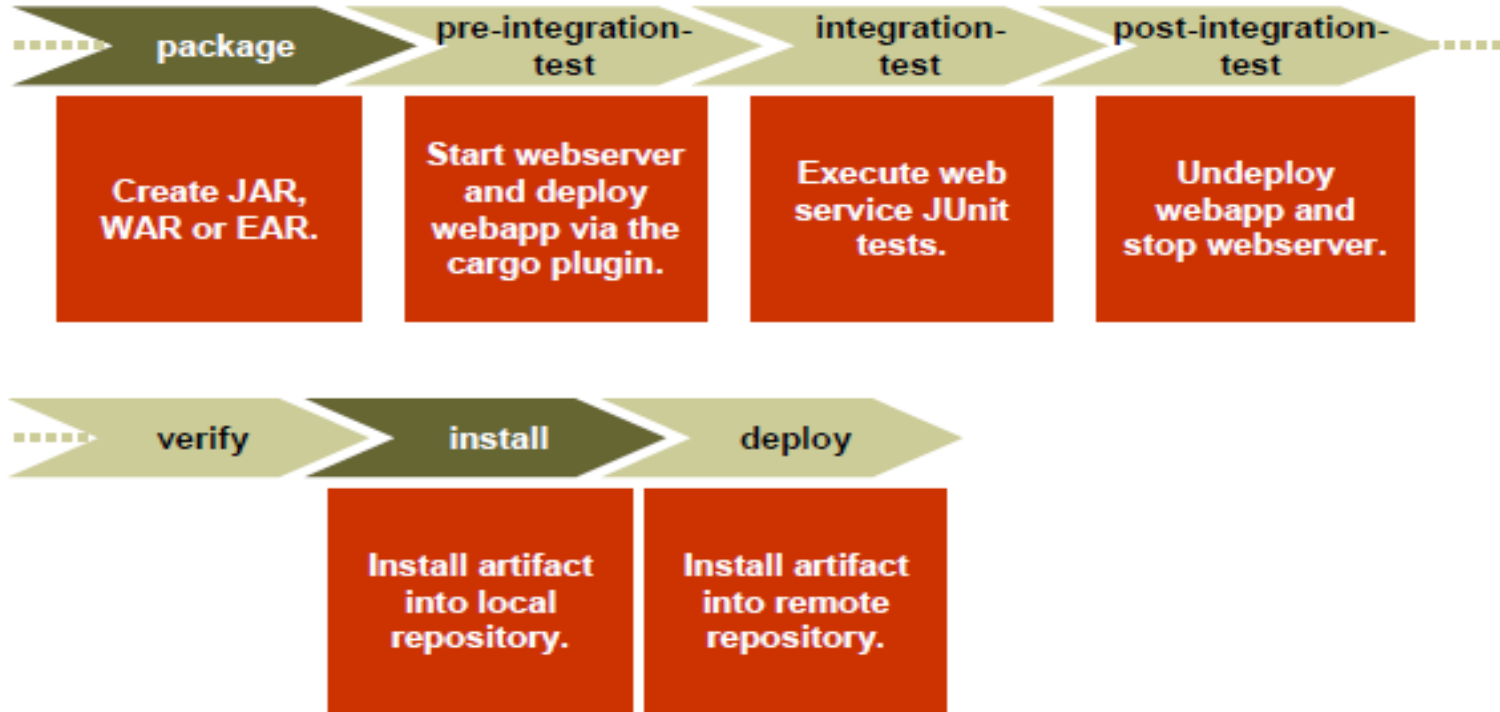
- A plugin is a set of goal or goals.
  - A goal is a “task” in Maven.
  - Goals can be executed independently
  - Goals can be executed as a part of a chain of goals.
  - Goals can have parameters with default values
  - Goal(s) can be attached to a Maven Life cycle Phase
- Goals are executed based on input found in pom.xml of that project
  - e.g., compiler:testCompile goal checks the input found in POM.xml for relevant parameters.

# Life Cycle – “Default”

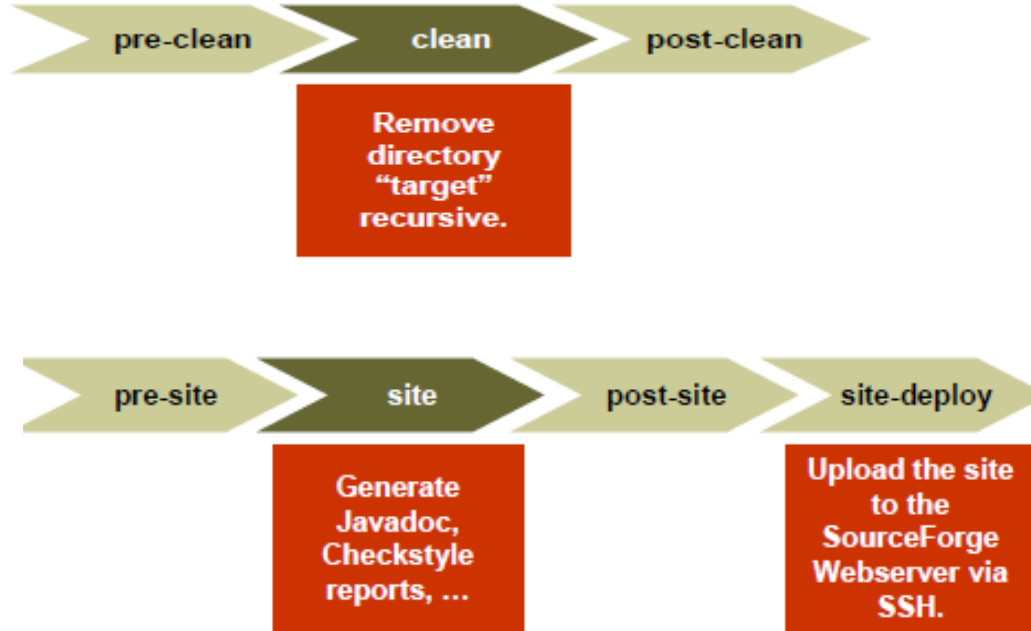
## lifecycle “default” (1)



# Life Cycle “default” (2)



# Lifecycle “clean” & “site”





# Maven life cycle

- Maven's default life cycle include the most common build phases like testing.
- The important Maven life cycle phases include the following list:
  - validate – It checks whether the project is correct & all information is available
  - compile – It compiles source code into binary artifacts
  - test – It executes the tests
  - package – It takes the compiled code & package it into jar/war/ear files
  - integration-test – It takes packaged result & executes additional tests, that requires the packaging
  - verify – It performs checks whether the package is valid
  - install – It installs the result of package phase into local Maven repository
  - deploy – It deploys the package to a local or remote repository

# Lab 4 : Exercise/Web Project



# Quiz

1) List of the proper hierarchy of Build lifecycle

- a) Verify
- b) Validate
- c) Compile
- d) Package
- e) Integration test
- f) Test
- g) Deploy
- h) Install

# Quiz

2) Command which generates jar file is

- a) Deploy
- b) Install
- c) Validate
- d) Package

3) Command which helps to release the build is

- a) Install
- b) Deploy
- c) Package
- d) Integration-test

# Site plugin

```
<plugin>  
<groupId>org.apache.maven.plugins</groupId>  
<artifactId>maven-site-plugin</artifactId>  
<version>3.7.1</version>  
</plugin>
```

```
<plugin>  
<groupId>org.apache.maven.plugins</groupId>  
<artifactId>maven-project-info-reports-plugin</artifactId>  
<version>3.0.0</version>  
</plugin>
```

# artifactory

1. Start & Login with admin credentials
2. Select => libs-release>set me up => generate settings>generate settings > Download snippet
  - Downloaded Settings.xml ( Decrypted credentials) place it within .m2 folder
  - Copy DistribuionManagement > update within POM
  - Update JDK compiler mapping (optional)
3. Define Artifactory server with credentials within maven settings.xml

```
<server>
  <username>devops</username>
  <password>devops</password>
  <id>central</id>
</server>
```

- Configure jenkins>Manage Jenkins> Configure System>Artifactory
- Test Connection (By pass proxy)

# webserver

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.1</version>
  <configuration>
    <url>http://localhost:5050/manager/text</url>
    <server>myserver</server>
    <path>/ILP_Bookstore</path>
    <update>true</update>
    <username>admin</username>
    <password>admin</password>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-deploy-plugin</artifactId>
  <version>2.7</version>
</plugin>
```

# Integration Test

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.9.1</version>
  <executions>
    <execution>
      <id>add-integration-test-sources</id>
      <phase>generate-test-sources</phase>
      <goals>
        <goal>add-test-source</goal>
      </goals>
      <configuration>
        <sources>
          <source>src/integration-test/java</source>
        </sources>
      </configuration>
    </execution>
    <execution>
      <id>add-integration-test-resources</id>
      <phase>generate-test-resources</phase>
      <goals>
        <goal>add-test-resource</goal>
      </goals>
      <configuration>
        <resources>
          <resource>
            <filtering>true</filtering>
            <directory>src/integration-test/resources</directory>
          </resource>
        </resources>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.18.1</version>
  <configuration>
    <excludedGroups>devops.ilpl.IntegrationTest</excludedGroups>
  </configuration>
</plugin>
```

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>2.19.1</version>
  <configuration>
    <includes>
      <include>/**/*.java</include>
    </includes>
    <groups>devops.ilpl.IntegrationTest</groups>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```



# Integration Test contd..

```
</plugin>
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <port>6080</port>
    <path>/ILP_Bookstore</path>
  </configuration>
  <executions>
    <execution>
      <id>start-tomcat</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <fork>true</fork>
      </configuration>
    </execution>
    <execution>
      <id>stop-tomcat</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>shutdown</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

# SonarQube

## ■ Maven settings.xml

```

<settings>
  <pluginGroups>
    <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
  </pluginGroups>
  <profiles>
    <profile>
      <id>sonar</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <!-- Optional URL to server. Default value is http://localhost:9000 -->
        <sonar.host.url>
          http://myserver:9000
        </sonar.host.url>
      </properties>
    </profile>
  </profiles>
</settings>

```

# Sonar scanner configuration

```
<plugin>  
  <groupId>org.sonarsource.scanner.maven</groupId>  
  <artifactId>sonar-maven-plugin</artifactId>  
  <version>3.6.0.1398</version>  
</plugin>
```

# References

1. <https://www.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/bowesjesse.pdf>



## Thank You

Avinash Patel

Senior Manager

[avinash.patel@wipro.com](mailto:avinash.patel@wipro.com)