Maven: Project management

The problem statement

- Large software contains tens/hundreds of projects & modules
- Distinct teams maintain distinct modules
- Hard to manage dependencies among modules
- Time consuming to build projects manually
  - *e.g. download each library*
  - *check dependency*

The solution

- We need to use **project management tool**
  - *Maven is one of the options*
- Helps with:
  - *Build process*
  - *Project structure*
  - *Dependency management*
  - *Access to information and documentation*

Maven

- Software project management and comprehension tool.
- Based on the concept of a project object model (POM).

- Manages project's build, reporting and documentation
  - from a central piece of information (pom.xml).

- The syntax for running Maven is as follows:
  - *mvn [options] [<goal(s)>] [<phase(s)>]*

The build process

- The Project Object Model (POM) is a configurations file
  - *It is the heart of Maven project*
  - *Contains project information and configuration details*
    - Dependencies
    - Commands to execute
    - Plugins
    - Metadata
  - *Used to build project*

Simple POM file example (pom.xml)

```
<project>
   <modelVersion>4.0.0</modelVersion>

   <groupId>edu.baylor.ecs.si</groupId>
   <artifactId>assignment-2</artifactId>
   <version>1.0-SNAPSHOT</version>

   <packaging>jar</packaging>
   <name>Assignment 2</name>

   <dependencies>
      <dependency>
          <groupId>commons-logging</groupId>
          <artifactId>commons-logging</artifactId>
          <version>1.1.1</version>
         <scope>compile</scope>
      </dependency>
  </dependencies>
</project>
```

Object model version

Group/organization id

Project identifier

Project version

Packaging type

Display name of project

Depe

Build lifecycle and phases

- The ***build lifecycle***
  - *is a process of building and distributing an artifact*
- A ***phase***
  - *a step in the build lifecycle*

Build lifecycle and phases

- Default phases:
  - *Validate*
  - *Compile*
  - *Test*
  - *Package*
  - *Install*
  - *Deploy*
- Optional phases
  - *Clean*
  - *Site*

Example goals

- To invoke a Maven build you set a lifecycle “goal”
- mvn install
  - *Invokes generate* and compile, test, package, integration-test, install*
- mvn clean
  - *Invokes just clean*
- mvn clean compile
  - *Clean old builds and execute generate*, compile*
- mvn compile install
  - *Invokes generate*, compile, test, integration-test, package, install*
- mvn test clean
  - *Invokes generate*, compile, test then cleans*

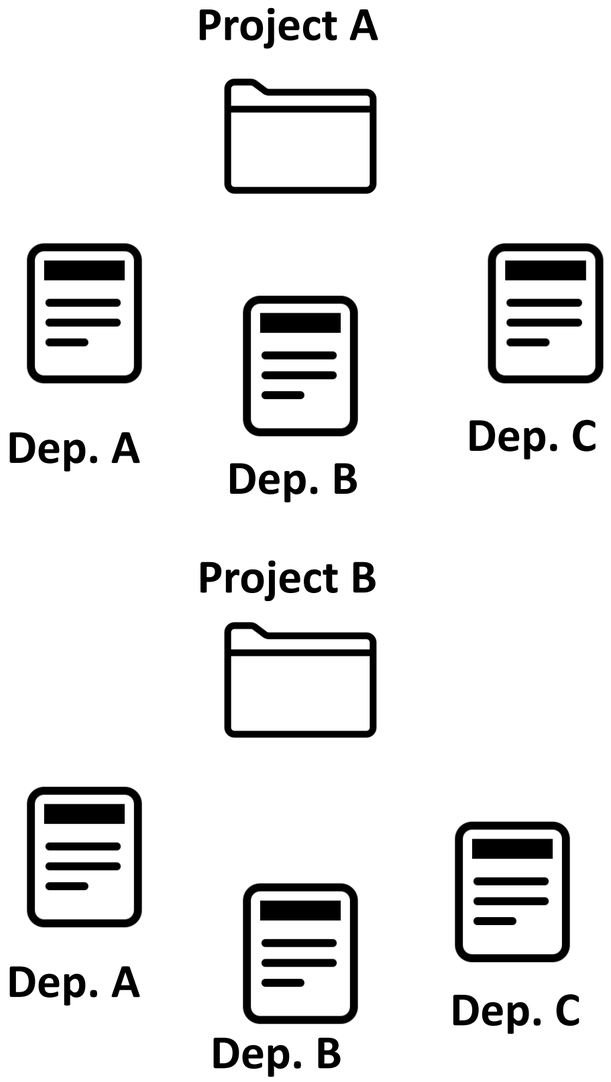
Standard directory layout

- Maven developer can get quickly familiar with new project
- No time wasted on directory reinvention
- src: All project source files go in this directory
- src/main: All sources that go into primary artifact
- src/test: All sources contributing to testing project
- src/main/java: All java source files
- src/main/webapp: All web source files
- src/main/resources: All non compiled source files
- src/test/java: All java test source files
- src/test/resources: All non compiled test source files

Maven and Dependencies

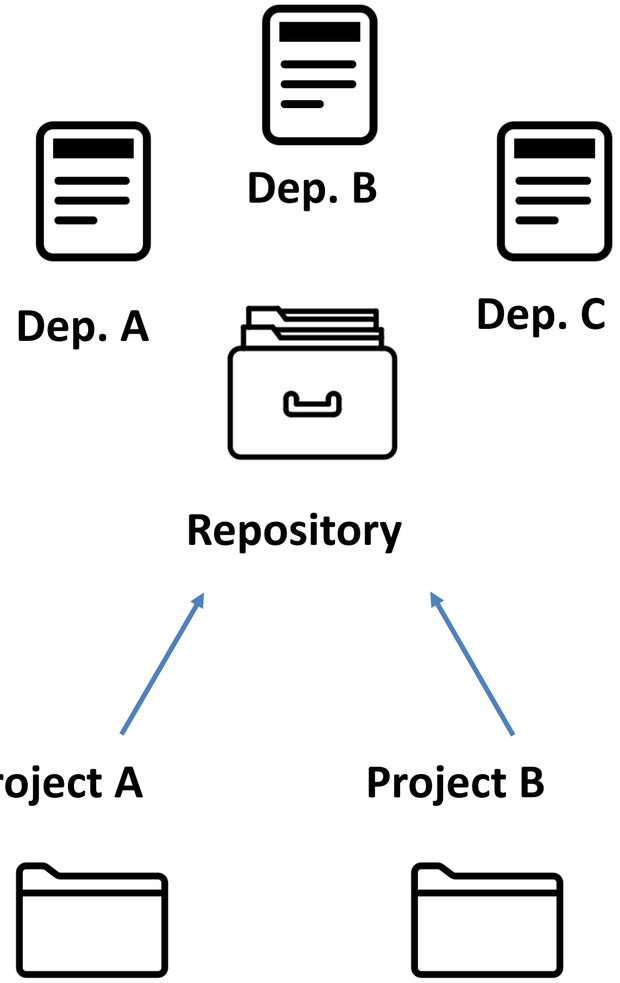
- Maven revolutionized Java dependency management
  - *No more checking libraries into version control (e.g. GIT)*
- Introduced the Maven Repository concept
  - *Established Maven Central*
- Created a module metadata file (POM)
- Introduced concept of transitive dependency
- Often include source and javadoc artifacts

Dependency illustration



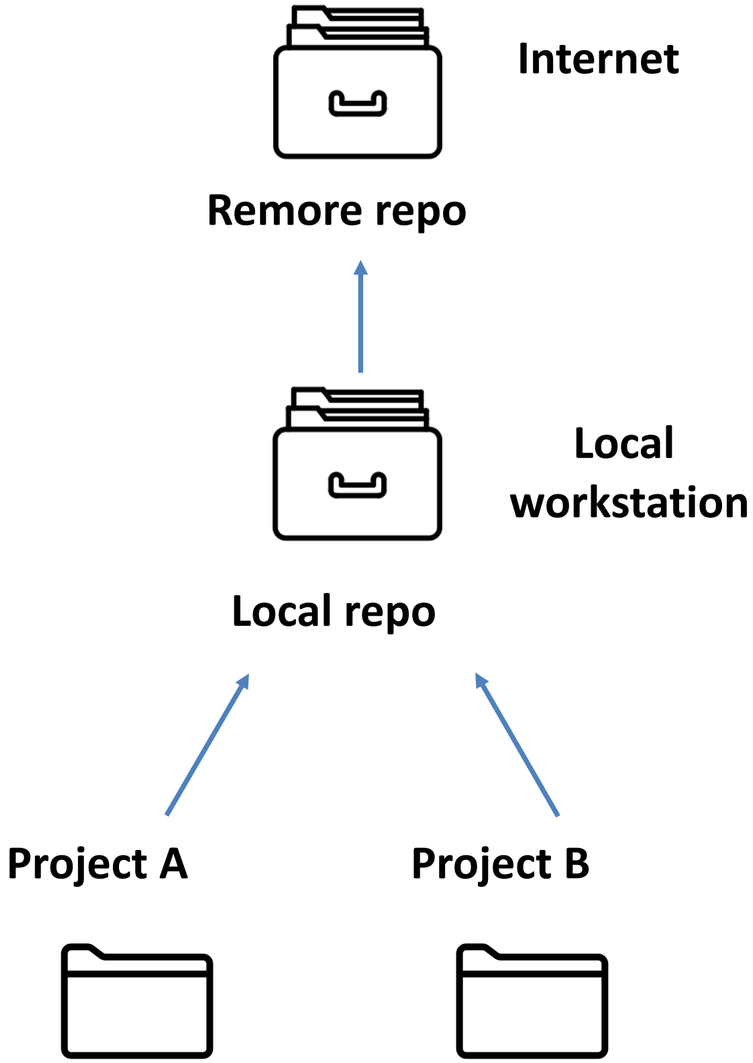
- **Ostrich solution**
  - *Replicate all dependencies per project*
  - *Occupy more storage!*
  - *Sharing project is slow*
  - *How to keep track of versions*

Dependency illustration



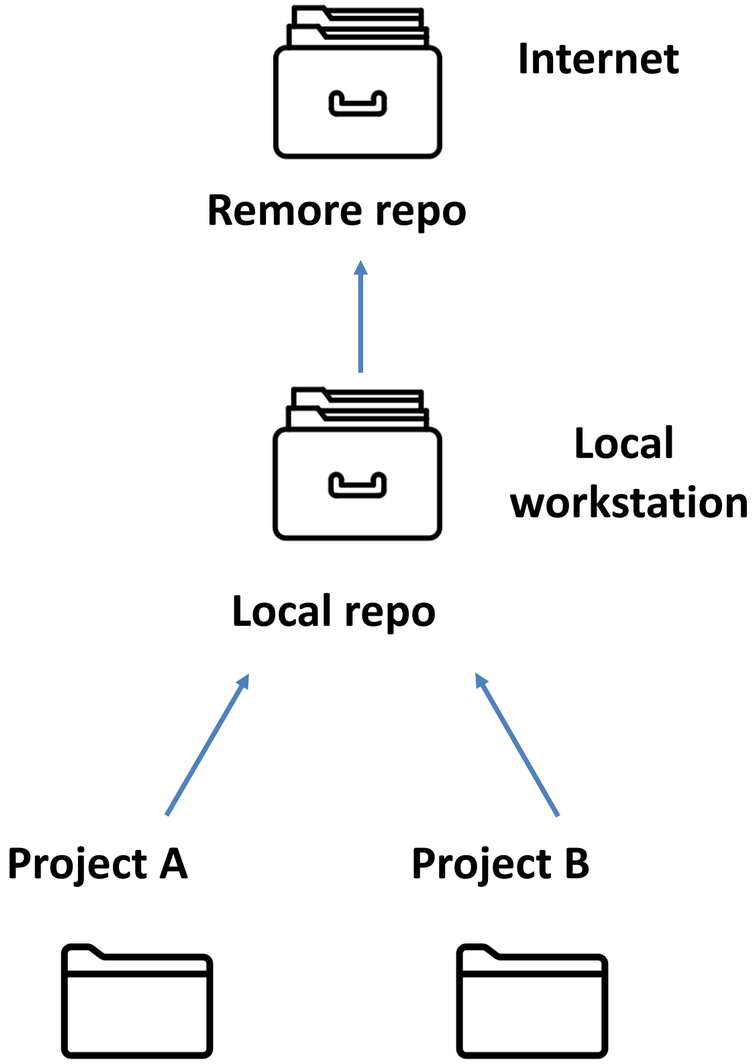
- **Preferred solution – Repository**
- Shared location for dependencies
  - *Single copy*
  - *Outside of the project*
  - *Linked through POM*

Dependency illustration



- Lesson from operating systems
  - ***Remote repository***
    - Provides artifacts to download
    - Maven central repository
  - ***Local repository***
    - Copy on local computer
    - Cache
    - Located at {user_home}/.m2/
    - Same as remote repository

Dependency illustration




```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.baylor.ecs.si</groupId>
  <artifactId>assignment-2</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Assignment 2</name>
   ..
  <repositories>
    <repository>
        <id>my-repo-</id>
       <url>http://my-server/repo
    </repository>
  </repositories>
   ..
</project>
```

Defining a repository

- Repositories are defined in the pom
- Repositories can be inherited from the parent
- Repositories are keyed by id
- Downloading snapshots can be controlled

```
<project>
    ...
    <repositories>
        <repository>
            <id>lds-main</id>
            <name>LDS Main Repo</name>
            <url>http://code.lds.org/nexus/content/groups/main-rep
        </repository>
    </repositories>
</project>
```

Defining a repository

- Repositories are defined in the pom
- Repositories can be inherited from parent
- Repositories are keyed by id
- Downloading snapshots can be controlled

```
<project>
    ...
    <repositories>
        <repository>
            <id>lds-main</id>
            <name>LDS Main Repo</name>
            <url>http://code.lds.org/nexus/content/groups/main-re
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

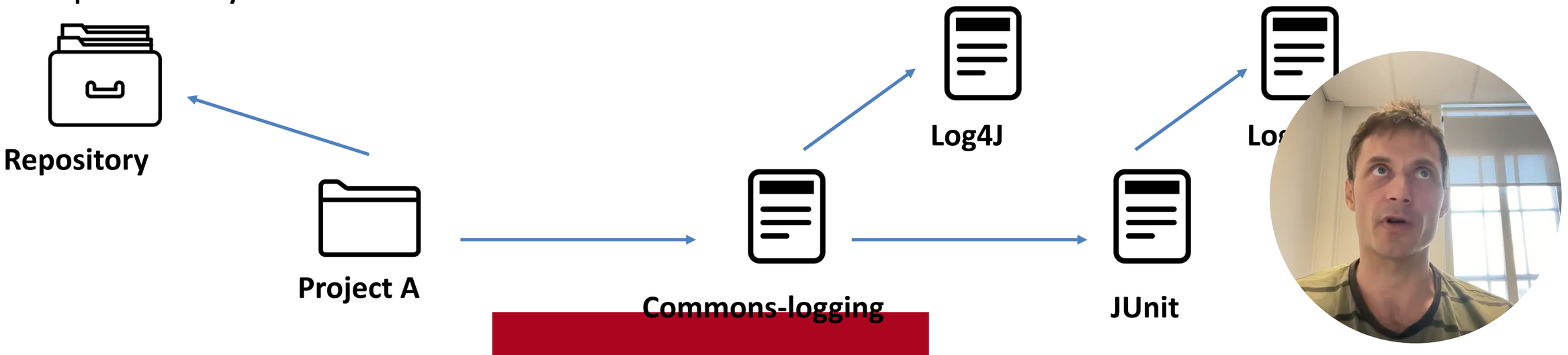
Adding a Dependency

- Dependencies consist of:
  - *Scope: compile, test, provided (default=compile)*
  - *Type: jar, pom, war, ear, zip (default=jar)*

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

Transitive Dependencies

- ProjectA depends on ProjectB, if ProjC depends on ProjectA then ProjectB is automatically included
- Only **compile** and **runtime** scopes are transitive
- Maven reads the POM files of your dependencies and automatically includes their required libraries
- No limit on the number of levels
- Dependency mediation – nearest definition



Dependency scope

```
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.4</version>
    <scope>compile</scope>
</dependency>
```

- Affects the classpath used for various build tasks
- Can be defined for all dependencies, compile default
- 5 dependency scopes available:
  - *Compile: Available in all classpaths (default)*
  - *Provided: The JDK or the container provides it*
  - *Runtime: Only required for execution, not for compilation*
  - *Test: Only required for testing, not for normal use (not deployed)*
  - *System: You provide it locally, not looked up in a repo*

Useful commands

$ mvn package	Compile and create JARs/WARs
$ mvn install	Package + copy to local repo
$ mvn clean	Delete target directory
$ mvn test	Run unit tests
$ mvn eclipse:eclipse	Create Eclipse project files
$ mvn idea:idea	Create IDEA project files
$ mvn jetty:run-war	Run a WAR file in Jetty server
$ mvn site	Generates project site
$ mvn install -DskipTests	Skip tests (saves time)

Summary

- Maven is a different kind of build tool
- It is easy to create multi-module builds
- Dependencies are awesome

Build a JAR

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
                <release>11</release>
            </configuration>
        </plugin>
        <!- see next page-->

</build>
```

Build a fat JAR

```
<build>
    <plugins>
        <!- see previous page -->
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descripto
                </descriptorRefs>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Build a fat JAR

```
<build>
	<plugins>
		<!- see previous page -->
		<plugin>
			<artifactId>maven-assembly-plugin</artifactId>
			<executions>
				<execution>
					<phase>package</phase>
					<goals>
						<goal>single</goal>
					</goals>
				</execution>
			</executions>
			<configuration>
				<descriptorRefs>
					<descriptorRef>jar-with-dependencies</descriptorRef>
				</descriptorRefs>
					<archive>
						<manifest>
							<addClasspath>true</addClasspath>
							<mainClass>softeng1.Main</mainClass>
						</manifest>
					</archive>
			</configuration>
		</plugin>
	</plugins>
</build>
```

Q/A

- …??? .. ?