Project Management
Software Projects

Composed of several modules

Use third-party libraries

Needs to be maintained

DEVELOPMENT IS NOT ONLY CODING
Development process

Dependency management
Building
Testing
Packaging
Deployment
Reports and metrics
Documentation
We need automation

Developers should care only of the configuration and then tools should perform the required actions

Automation saves time and makes the process easily reproducible
The maven approach

**maven**

- Fully automated project management tasks
- Pre-defined phases to support the development lifecycle
- Easily extensible via plug-ins to get more features
Build lifecycle

Composed of several *phases*

- Validate
- Compile
- Test
- Package
- Install
- Deploy

You are not going to miss something. Each phases requires the correct execution of the previous
Validate

Maven checks that the project is properly configured.

Correctness of the configuration file

Availability of everything needed by other phases
Compile

This phase only compiles all the java files in the project

It is useful to separate dependencies needed to build the project with others needed at later phases
Test

Executes tests and reports their outcome

Note: test failures will block the execution of other phases
Package

Creates a package (usually a jar) containing the compiled code and (if needed) other resources
Install

Copies the package to the local repository

Once the project is installed in a repository it can be used as a dependency
Deploy

Publishes the package to a remote repository so that they can made publicly available
Additional features

We can integrate plug-ins in the lifecycle adding more features.

For instance: documentation generation, reporting, metrics, ...
How it works

Based on conventions

Standard directory layout

Standard way to manage all the projects

Everything is configured in a

**Project Object Model**

(POM) file
Project layout

- Project code
- Project test suite
- Non-code resources
- Packages, intermediate files, ...
- Configuration file
A simple POM file

  <modelVersion>4.0.0</modelVersion>

  <groupId>it.polimi.dei</groupId>
  <artifactId>esercitazioniIngegneriaDelSoftware</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>esercitazioniIngegneriaDelSoftware</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
Dependency management

**traditional way**
- Search for the library
- Download and configure the required dependencies
- Set-up everything again for runtime

**with maven**
- Tell maven what you need
Example

We want to add Google guava library

Everything we need to do is to add this lines to the POM!

```xml
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>11.0</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
```
Example

What if you want to use a library just for testing?

Everything we need to do is to add this lines to the POM!

```xml
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.8.1</version>
  <scope>test</scope>
</dependency>
```
Dependency management

- Support for multiple library versions
- System repository to avoid library duplication
- Remote repository with a lot of software available
Building

No need to manually invoke the compiler

Projects are build accordingly to the information in the POM

Maven know where to find source files, libraries and how to build the application
Testing

Crucial to ensure that the application is working properly

Maven integrates with jUnit

Every time the application is packaged, the test suite is run
Packaging

The compiled files are packaged in a single archive

Maven supports several packages: jar, war, ...

Package is available in the target directory
Reporting

Makes it easier to generate reports for the projects

API Documentation

Test results and coverage

Code quality

Everything can be made available on a website
Useful commands

- **package**: Compilation, testing and JAR generation
- **install**: Packaging and copy to the local repository
- **test**: Unit testing
- **clean**: Deletes target directory
- **site**: Generates project site