

Technological foundations of software development²

Automate build production

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

Objectives of the session

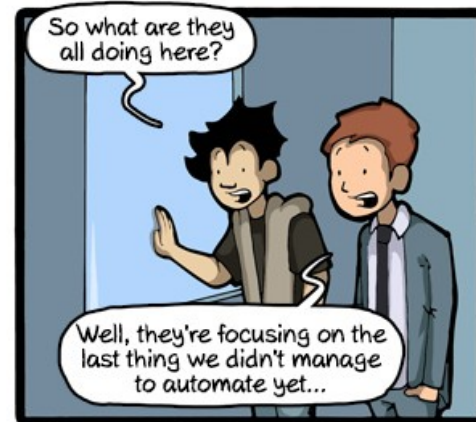
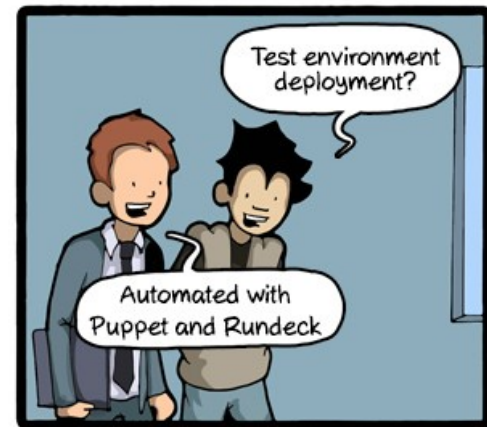
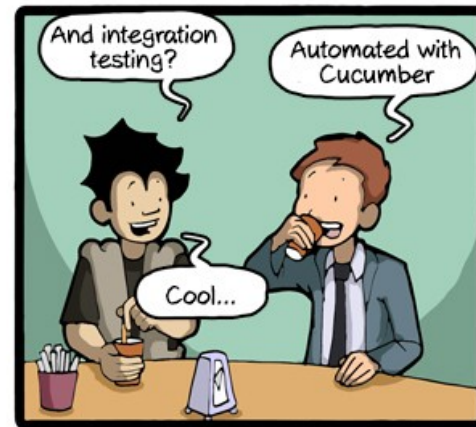
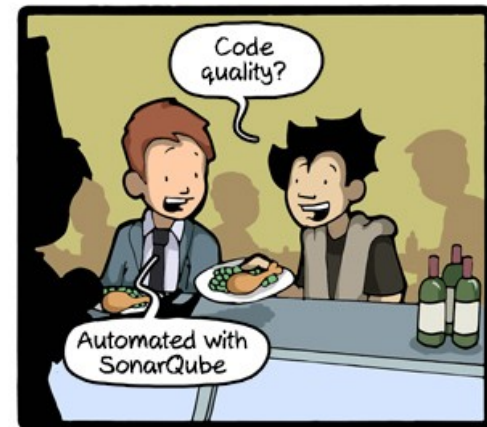
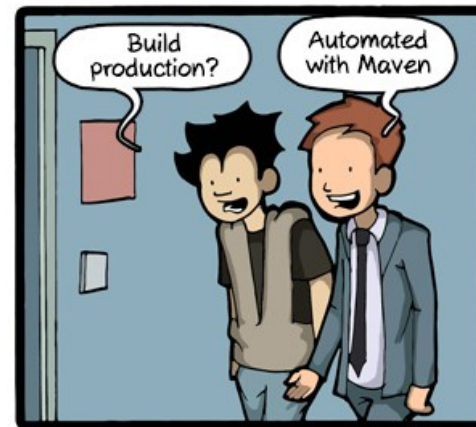
This session aims to familiarize you with the methods and tools for automating code production: compilation, testing, packaging, deployment, etc. In particular, we will see:

- make
- Apache Maven

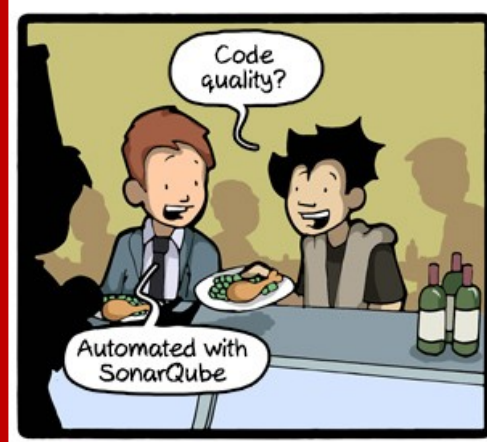
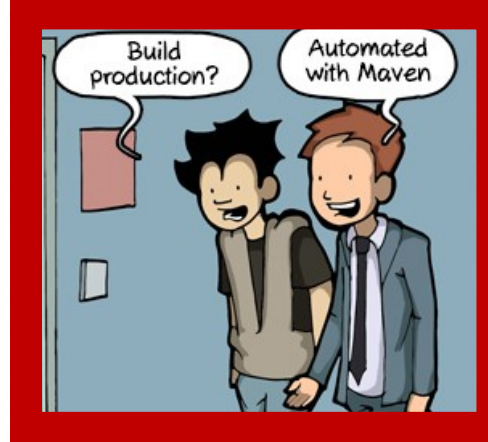
Technological foundations of software development

Automate build production

Part 1: « Automate »: What? Why? How?



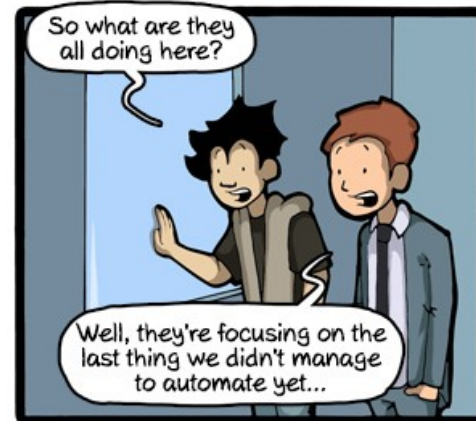
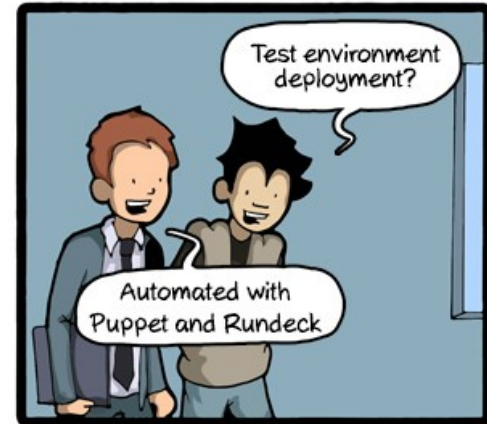
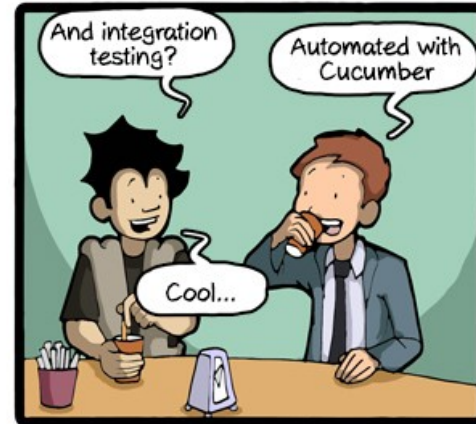
this session
Build Production



Technological foundations of software development

Automate build production

Part 1: « Automate »: What? Why? How?



CommitStrip.com

« Automate »: What ?

- Check the licenses of the files, check that the remote repository has no new commit, ...
- Potentially generate source code
- Download or update dependencies
- Manage additional resources
- Compile sources, optimize code
- Run unit tests
- Generate documentation
- Package executable code
- Deploy in a test environment and execute integration tests
- Verify the integrity of the archive, check the quality of the code, ...
- Deploy in a production environment, publish the code version
- Create and push a git tag
- ...

« Automate »: Why ?

- Accelerate software production
- Improve software quality
- Avoid redundant tasks
- Limit bad software versions
- History: traceability, non-repudiation
- Save time and money
- As a **building block** for continuous integration and deployment

« Automate »: How ?

- Build automation utilities
 - examples: make, rake, msbuild, ant, maven, gradle, webpack, ...
 - automate simple and repeatable tasks
 - order tasks to achieve goals
 - execute only the necessary tasks
- Two paradigms:
 - task-oriented: breaks down goals into tasks
 - product-oriented: breaks down into sub-products to be generated

« Automate »: How ?

- **Build automation servers**

- run build automation utilities

- Three paradigms:

- **On-demand automation:** the user requests the execution of the production
 - **Scheduled automation:** execution is scheduled (e.g. nightly build)
 - **Triggered automation:** execution is triggered by an event (e.g., commit on master)

Technological foundations of software development

Automate build production

Part 2: Build automation utilities

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

Technological foundations of software development

Automate build production

Part 2: Build automation utilities

Part 2.1: *Make-like*

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

Make-like build automation utilities

- macros, declarative programming
- first implementation by Stuart Feldman (Bell Labs, released in 1976)
- many variants of the tool
 - make, GNU gmake, Microsoft nmake (in Visual Studio), google Kati (for Android OS) ...
- still widely used today

file makefile (or Makefile, or GNUmakefile, ...)

```
srcfiles := $(shell echo src/{00..99}.txt)
destfiles := $(patsubst src/%.txt,dest/%.txt,$(srcfiles))

tutorial:
    @echo "10 questions about Isaac's Makefile in the MCQ"

src/%.txt:
    @[ -d src ] || mkdir src
    echo $* > $@

dest/%.txt: src/%.txt
    @[ -d dest ] || mkdir dest
    cp $< $@

destination: $(destfiles)

.PHONY: tutorial destination
```

```
$ make          # calls rule "tutorial"
$ make dest/00.txt # calls third rule, which needs second.
$ make destination # updates all targets
```

Macros: reusable pieces of text or values that can be substituted throughout the build

Declarative programming rules

target [target ...]: [component ...]
[Tab ↹] [command 1]

...
[Tab ↹] [command n]

"tutorial" and "destination" aren't actual filename, so we define them as .PHONY

call with target file names

To read for MCQ test

- “Isaac’s Makefile”
<https://gist.github.com/isaacs/62a2d1825d04437c6f08>

Technological foundations of software development

Automate build production

Part 1: Build automation utilities

Part 2.2: for Java

ICM – Computer Science Major – Course unit on Technological foundations of computer science

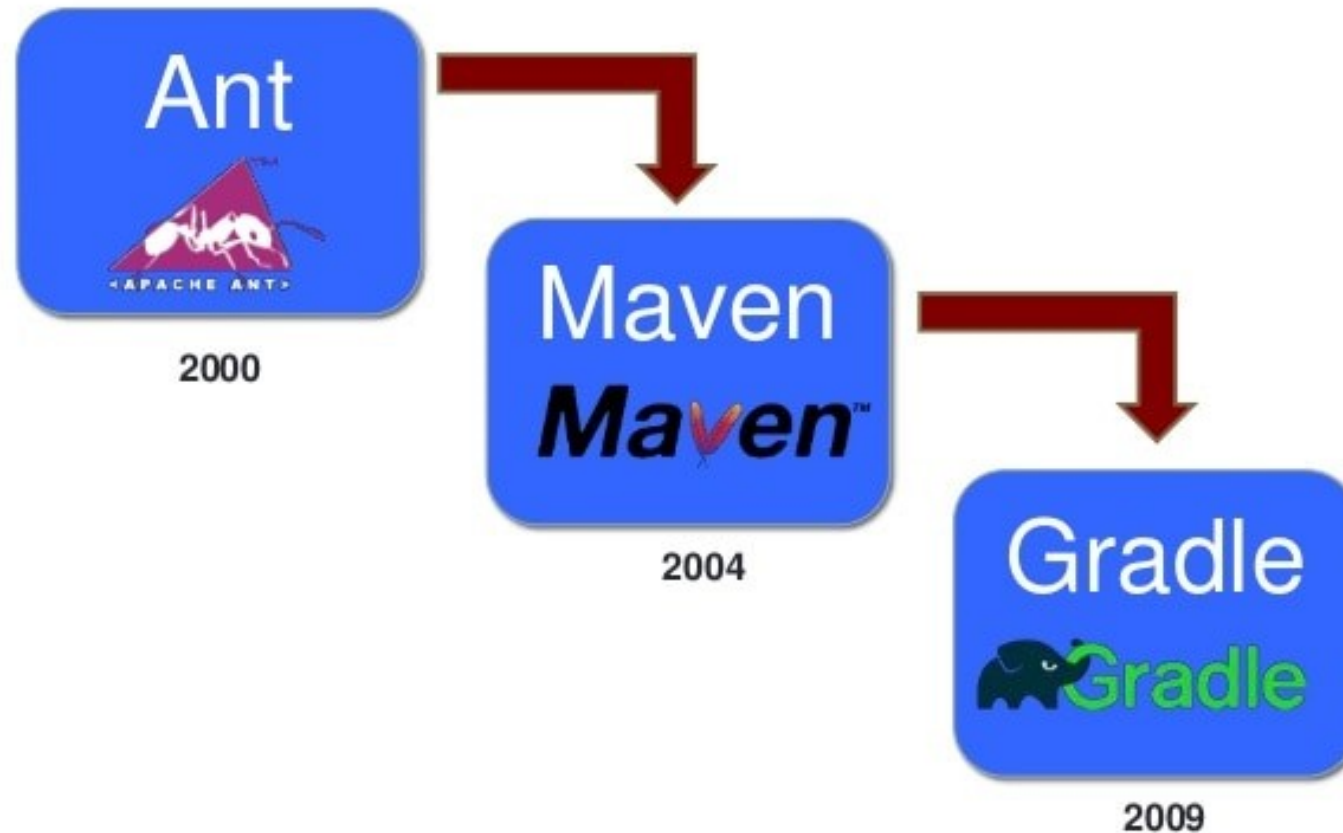
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

For Java

Build System Evolution



For Java



Apache Ant ("Another Neat Tool")

Apache Software Foundation, v1 2000 ;
written in **Java**

XML project files, complex, verbose

https://en.wikipedia.org/wiki/Apache_Ant

```
1 <project name="HelloWorldBuildTest" basedir="." default="main">
2   <property name="src.dir" value="src"/>
3   <property name="build.dir" value="build"/>
4   <property name="classes.dir" value="{build.dir}/classes"/>
5   <property name="jar.dir" value="{build.dir}/jar"/>
6   <property name="lib.dir" value="lib"/>
7   <property name="main-class" value="package.HelloWorld"/>
8
9   <path id="classpath">
10    <fileset dir="{lib.dir}" includes="**/*.jar"/>
11  </path>
12  <target name="clean">
13    <delete dir="{build.dir}"/>
14  </target>
15  <target name="compile">
16    <mkdir dir="{classes.dir}"/>
17    <javac srcdir="{src.dir}" destdir="{classes.dir}" classpathref="classpath"/>
18  </target>
19  <target name="jar" depends="compile">
20    <mkdir dir="{jar.dir}"/>
21    <jar destfile="{jar.dir}/{ant.project.name}.jar" basedir="{classes.dir}">
22      <manifest>
23        <attribute name="Main-Class" value="{main-class}"/>
24      </manifest>
25    </jar>
26  </target>
27  <target name="run" depends="jar">
28    <java classname="{main-class}">
```



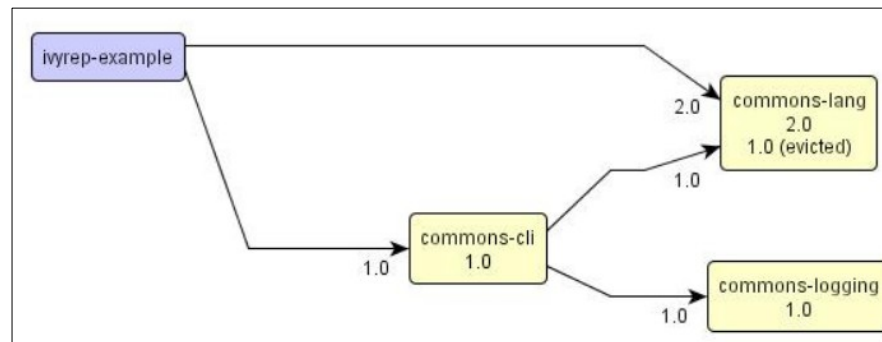
<http://blog.travelmarx.com/2011/10/java-apache-ant-and-hello-world.html>

+ Apache Ivy

Transitive dependency manager
example:

```
<dependency org="com.google.code.gson" name="gson" rev="2.8.8"/>
```

<http://ant.apache.org/ivy/>



<http://www.java.free.fr/ivy/doc/print.html>

For Java

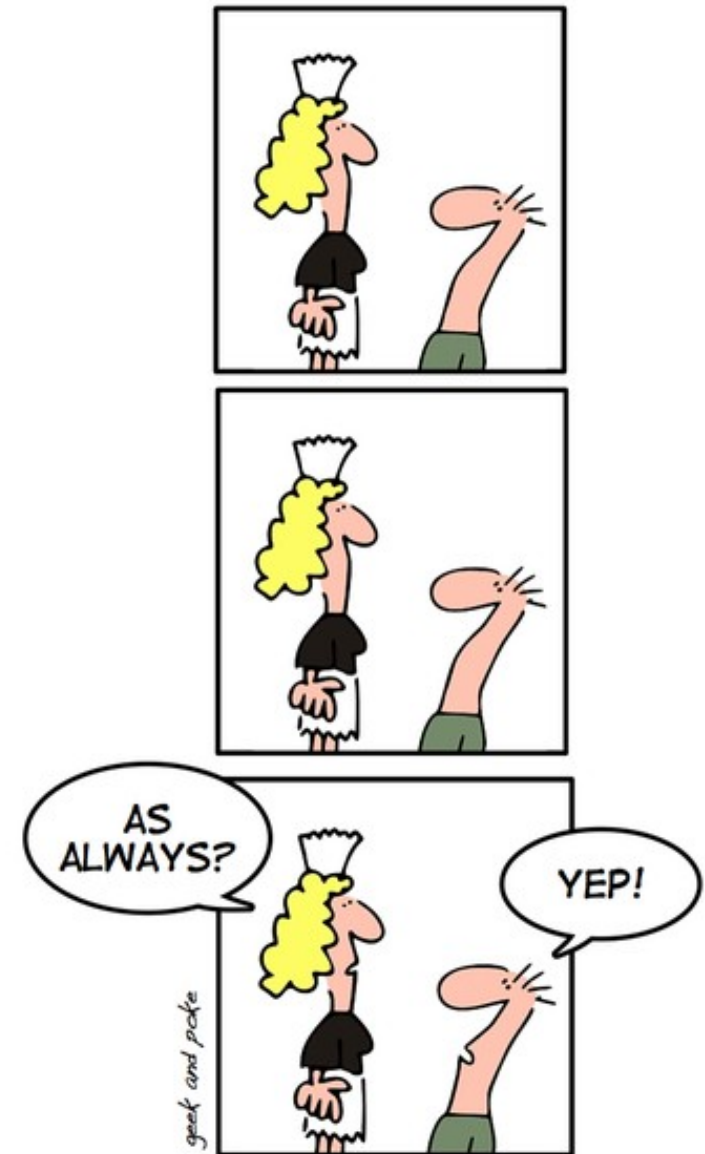
Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

objective: limit the number of decisions a developer has to make.



**SIMPLY EXPLAINED - PART 18:
CONVENTION OVER CONFIGURATION**

For Java

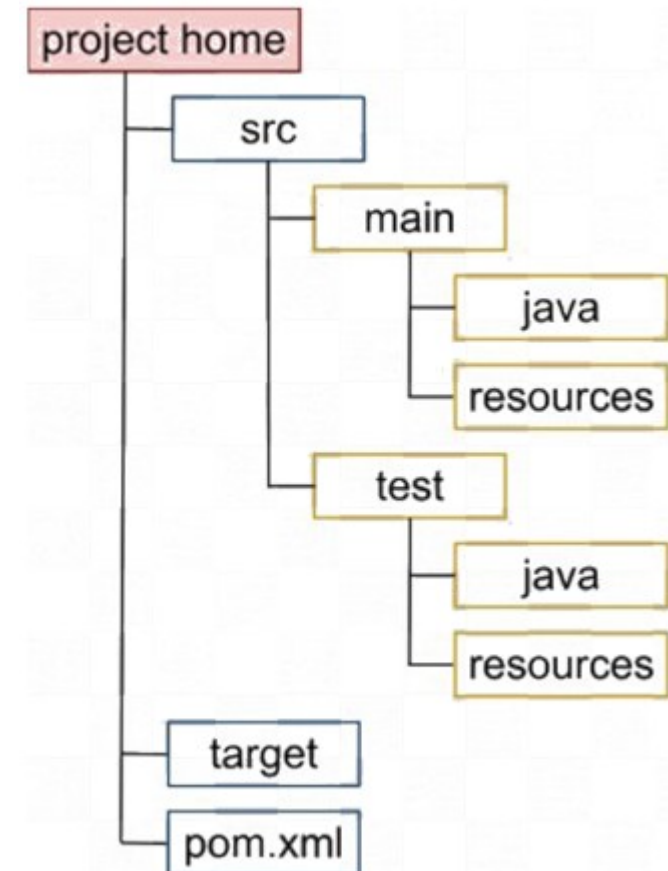
Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

- Conventions for the project structure



For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

- Conventions for the project structure
- A project is described by an XML file:
le **POM (Project Object Model)**

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <!-- The Basics -->
6.   <groupId>...</groupId>
7.   <artifactId>...</artifactId>
8.   <version>...</version>
9.   <packaging>...</packaging>
10.  <dependencies>...</dependencies>
11.  <parent>...</parent>
12.  <dependencyManagement>...</dependencyManagement>
13.  <modules>...</modules>
14.  <properties>...</properties>
15.
16.  <!-- Build Settings -->
17.  <build>...</build>
18.  <reporting>...</reporting>
19.
20.  <!-- More Project Information -->
21.  <name>...</name>
22.  <description>...</description>
23.  <url>...</url>
24.  <inceptionYear>...</inceptionYear>
25.  <licenses>...</licenses>
26.  <organization>...</organization>
27.  <developers>...</developers>
28.  <contributors>...</contributors>
29.
30.  <!-- Environment Settings -->
31.  <issueManagement>...</issueManagement>
32.  <ciManagement>...</ciManagement>
33.  <mailingLists>...</mailingLists>
34.  <scm>...</scm>
35.  <prerequisites>...</prerequisites>
36.  <repositories>...</repositories>
37.  <pluginRepositories>...</pluginRepositories>
38.  <distributionManagement>...</distributionManagement>
39.  <profiles>...</profiles>
40. </project>
```

For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

- Conventions for the project structure
- A project is described by an XML file:
le **POM (Project Object Model)**

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <!-- The Basics -->
6.   <groupId>...</groupId>
7.   <artifactId>...</artifactId>
8.   <version>...</version>
9.   <packaging>...</packaging>
10.  <dependencies>...</dependencies>
11.  <parent>...</parent>
12.  <dependencyManagement>...</dependencyManagement>
13.  <modules>...</modules>
14.  <properties>...</properties>
15.
16.  <!-- Build Settings -->
17.  <build>...</build>
18.  <reporting>...</reporting>
19.
20.  <!-- More Project Information -->
21.  <name>...</name>
22.  <description>...</description>
23.  <url>...</url>
24.  <inceptionYear>...</inceptionYear>
25.  <licenses>...</licenses>
26.  <organization>...</organization>
27.  <developers>...</developers>
28.  <contributors>...</contributors>
29.
30.  <!-- Environment Settings -->
31.  <issueManagement>...</issueManagement>
32.  <ciManagement>...</ciManagement>
33.  <mailingLists>...</mailingLists>
34.  <scm>...</scm>
35.  <prerequisites>...</prerequisites>
36.  <repositories>...</repositories>
37.  <pluginRepositories>...</pluginRepositories>
38.  <distributionManagement>...</distributionManagement>
39.  <profiles>...</profiles>
40. </project>
```

G-A-V (groupId, artifactId, version) is indicated for lines 6-8.

Annotations and arrows point to specific XML elements:

- jar (default), war, pom, ...** points to line 9 (`<packaging>`).
- dependencies** points to line 10 (`<dependencies>`).
- inheritance and multi-modules** points to line 11 (`<parent>`) and line 12 (`<dependencyManagement>`).
- configuration `${prop}`** points to line 14 (`<properties>`).
- plugins used during the build** points to line 17 (`<build>`).
- plugins for reporting** points to line 18 (`<reporting>`).
- metadata about the projet** points to lines 21-28 (project information section).
- config. environnement** points to lines 31-39 (environment settings section).

For Java

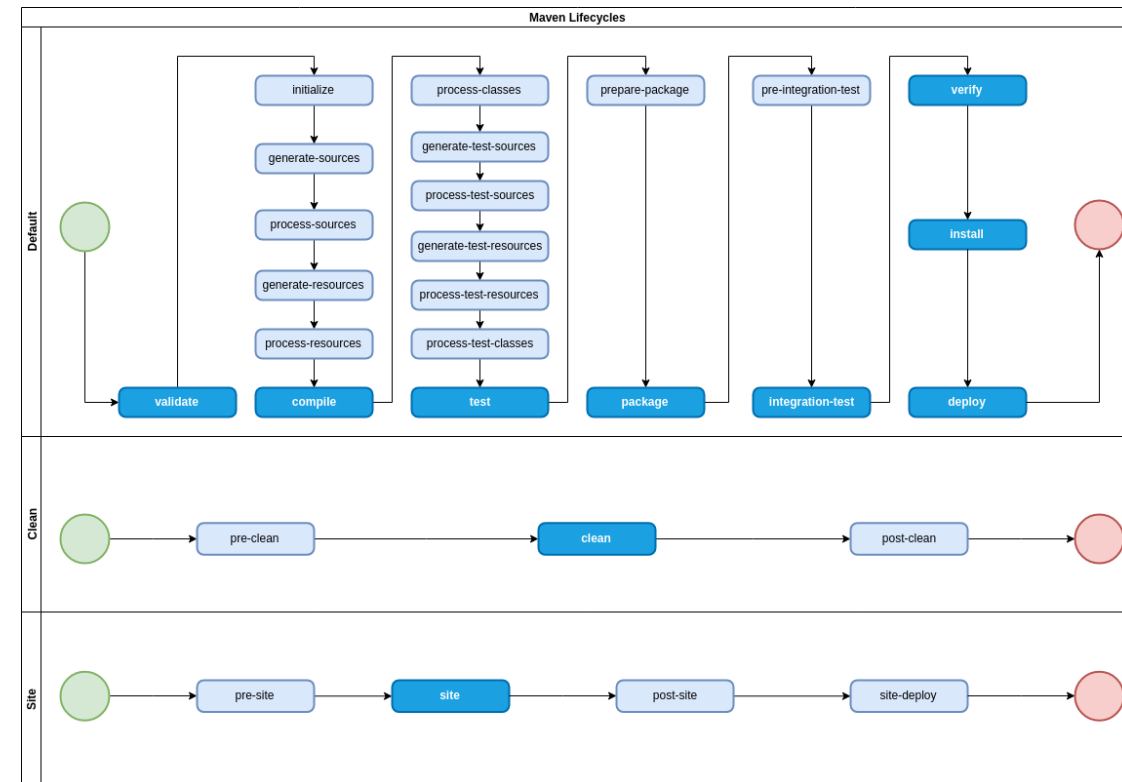
Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

- Conventions for the project structure
- A project is described by an XML file:
le **POM** (**P**roject **O**bject **M**odel)
- **Standardized life cycle (phases)**



<https://medium.com/@yetanothersoftwareengineer/maven-lifecycle-phases-plugins-and-goals-25d8e33fa22>

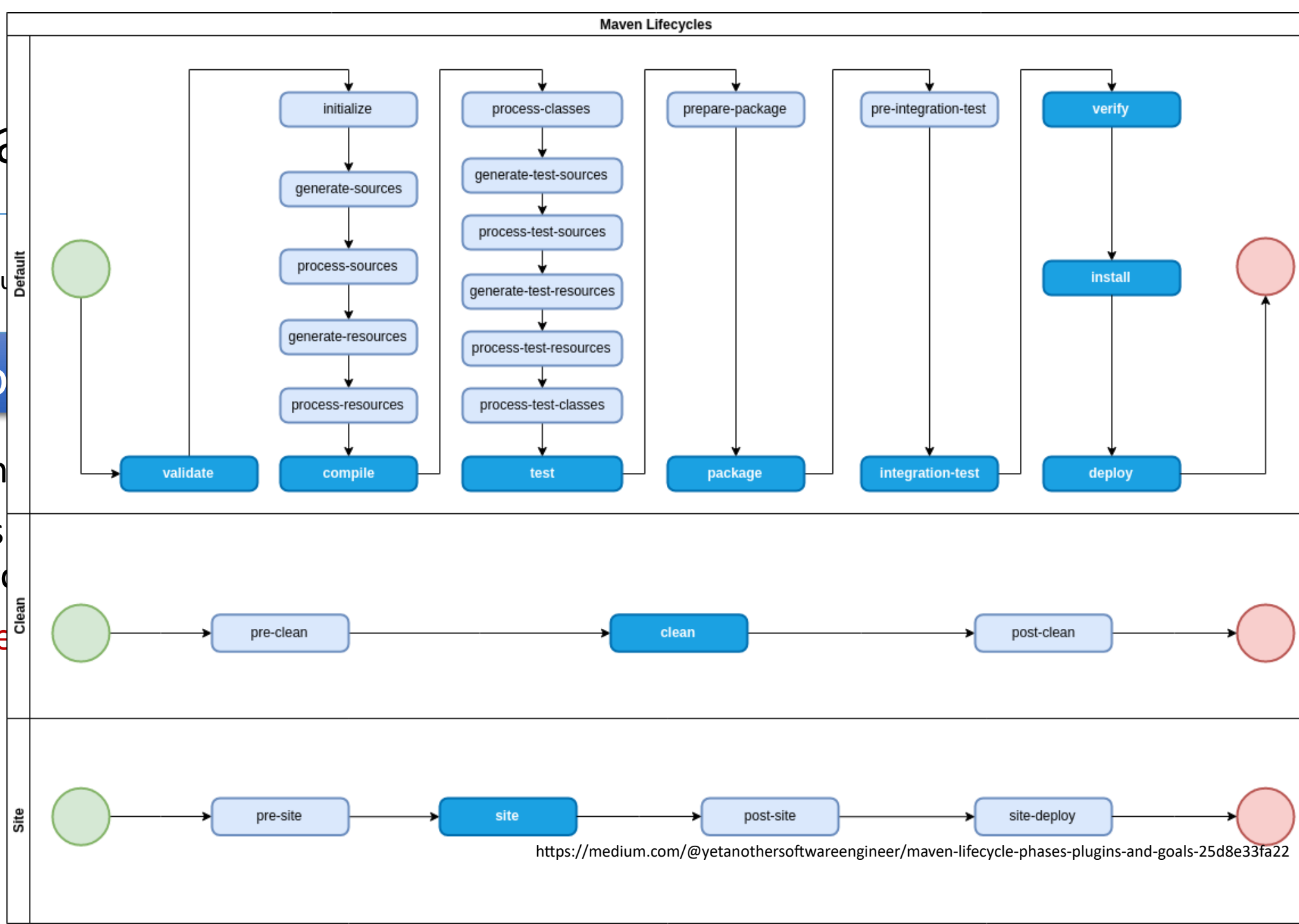
For Java

Apache Maven

Apache Software Foundation
written in Java

Convention

- Convention
- A project is described by a file called **POM** (Project Object Model)
- **Standardized**



<https://medium.com/@yetanothersoftwareengineer/maven-lifecycle-phases-plugins-and-goals-25d8e33fa22>

For Java

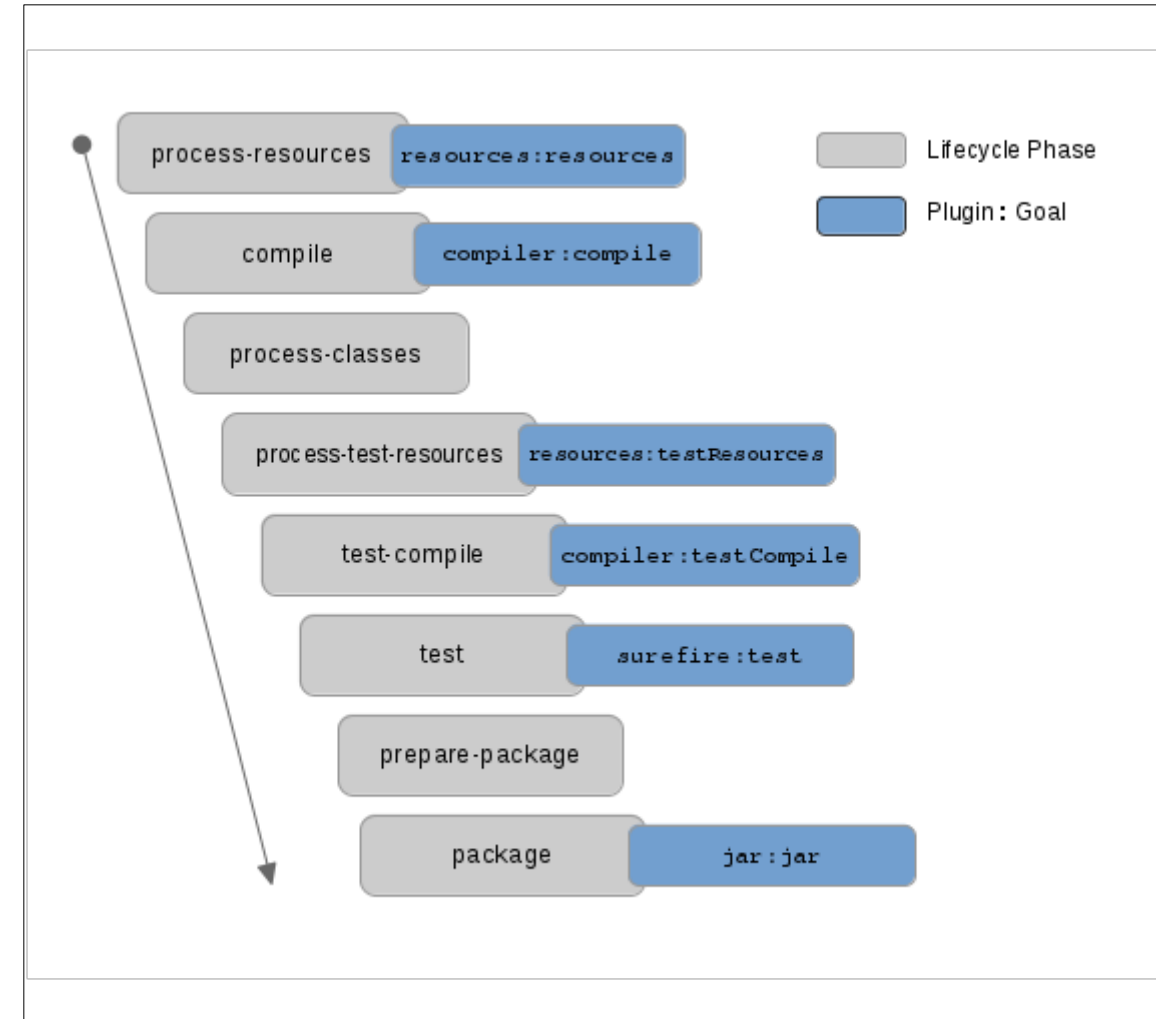
Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Convention over configuration

- Conventions for the project structure
- A project is described by an XML file:
le **POM** (**P**roject **O**bject **M**odel)
- Standardized life cycle (phases)
- Convention for **plugin goals** executed
at each phase of the lifecycle



Example for `<packaging>jar</packaging>`

For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Dependency management

- Use G-A-V coordinates
- concepts:
 - **repositories**: where dependencies are downloaded to
 - **scope**: context of use of the dependency
 - **transitivity**: dependencies of dependencies
 - **inheritance**: inheritance of dependencies from parent project

```
1. <project xmlns="http://maven.apache.org/POM/
2.   xsi:schemaLocation="http://maven.apache.or
3.   ...
4.   <dependencies>
5.     <dependency>
6.       <groupId>junit</groupId>
7.       <artifactId>junit</artifactId>
8.       <version>4.12</version>
9.       <type>jar</type>
10.      <scope>test</scope>
11.      <optional>true</optional>
12.    </dependency>
13.    ...
14.  </dependencies>
15.  ...
16. </project>
```

<https://maven.apache.org/pom.html#dependencies>

For Java

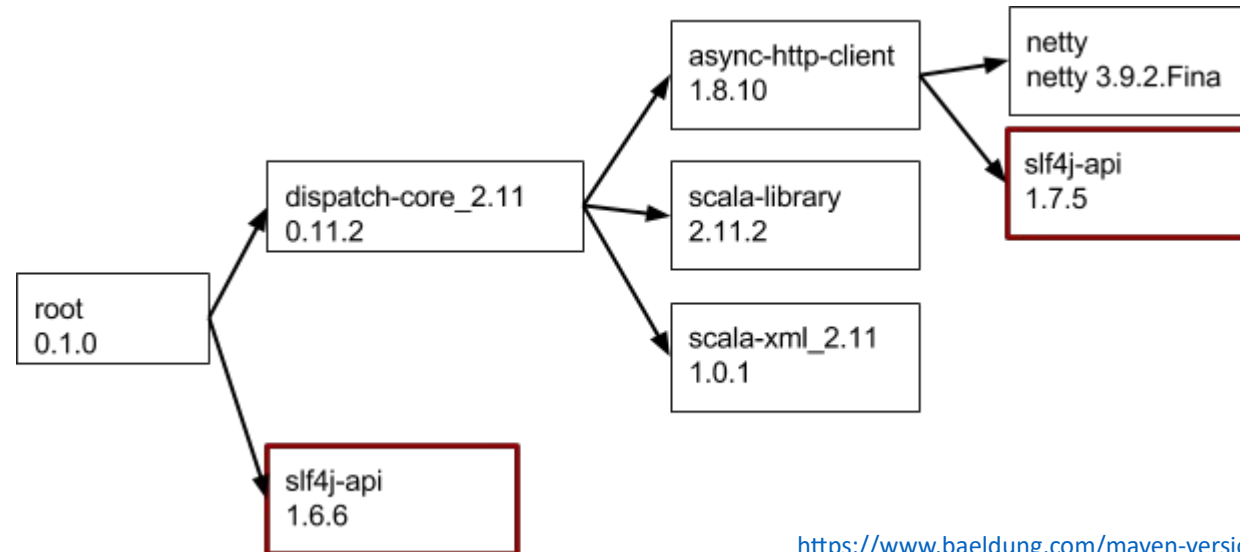
Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Dependency management

- Use G-A-V coordinates
- ⚠ dependence on multiple versions of the same artifact?



For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Dependency management

- Use G-A-V coordinates
- ⚠ dependence on multiple versions of the same artifact?

➤ Flexibility in version numbers :

- [1.0,) : version 1.0 or greater
- (,1.0] : version lower or equal to 1.0
- [1.0,1.2] : between versions 1.0 and 1.2, inclusive
- (,1.2),(1.2,) : all versions except 1.2
- [1.0,2.0) : version greater or equal to 1.0 and lower than 2.0

➤ Other solutions: <https://www.baeldung.com/maven-version-collision>

For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



Everything is plugin

- A plugin is a jar that contains a class annotated @Mojo
- `<G>:<A>:<V>:<goal>`
 - example: `org.apache.maven.plugins:maven-compiler-plugin:3.8.1:compile`
- Convention for plugin goals executed at each phase of the lifecycle
- Configuration of other plugins, ...
 - Many plugins already published by Maven <https://maven.apache.org/plugins/>
 - Example: <https://maven.apache.org/plugins/maven-deploy-plugin/examples/deploy-ftp.html>
 - Convention for the phases to which a goal is attached

For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



- **Execution** of a life cycle goal

- Executes all goals associated with all phases $\leq$ package

```
$ mvn package
```

- **Execution** of a specific goal of a specific plugin

- Executes goal effective-pom from plugin help

```
$ mvn help:effective-pom
```

```
$ mvn org.apache.maven.plugins:maven-help-plugin:3.2.0:effective-pom
```

- Passing parameters (same as in POM `<parameters>...</parameters>`)

- Executes goal describe from plugin help

```
$ mvn help:describe -Dplugin=help -Dminimal
```

For Java

Apache Maven

Apache Software Foundation, v1 2004; v2 2005; v3 2013
written in **Java**



- pointers:
 - @fr <http://www.jmdoudoux.fr/java/dej/indexavecframes.htm>
chapter 93. Maven (except: 93.2, 93.3.6, 93.3.7, 93.3.9)
 - <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
 - <https://maven.apache.org/guides/getting-started/index.html>

For Java

Apache Gradle

Open Source, 2007
written in Java, Groovy, Kotlin



Convention over configuration

- Domain specific language (DSL) rather than XML
 - DSL based on Groovy, or on Kotlin
 - More expressive, concise, flexible, than Ant and Maven
- Acyclic oriented graph of tasks
 - execution of tasks in parallel or in sequence
 - dependencies between tasks
 - incremental production

```

<?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">
  <modelVersion>4.0.0</modelVersion>

  <groupId>fr.mines_stetienne.ci.i2si</groupId>
  <artifactId>calculator</artifactId>
  <version>0.8.32</version>

  <name>Calculator</name>
  <description>The example Calculator maven project</description>
  <inceptionYear>2019</inceptionYear>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <organization>
    <name>École des Mines de Saint-Étienne</name>
    <url>http://www.mines-stetienne.fr</url>
  </organization>

  <licenses>
    <license>
      <name>The Apache Software License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
    </license>
  </licenses>

  <dependencies>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.25</version>
  </dependency>
    <dependency>
      <groupId>commons-cli</groupId>
      <artifactId>commons-cli</artifactId>
      <version>1.4</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>

```

pom.xml

```

apply plugin: 'java'
apply plugin: 'maven'

```

```

group = 'fr.mines_stetienne.ci.i2si'
version = '0.8.32'

```

```

description = """"Calculator""""

```

```

sourceCompatibility = 1.5
targetCompatibility = 1.5

```

```

repositories {
    maven { url "https://repo.maven.apache.org/maven2" }
}
dependencies {
    compile group: 'org.slf4j', name: 'slf4j-log4j12', version: '1.7.25'
    compile group: 'commons-cli', name: 'commons-cli', version: '1.4'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

```

build.gradle

settings.gradle

```

rootProject.name = 'calculator'

```



maven



For Java

Apache Gradle

Open Source, 2007
written in Java, Groovy, Kotlin



- Viewed in details in the course “Web Programming”
- pointers:
 - https://docs.gradle.org/current/userguide/what_is_gradle.html
 - <https://spring.io/guides/gs/gradle/>
 - https://dev-mind.fr/training/gradle/gradle_en.html

Technological foundations of software development

Automate build production

Part 1: Build automation utilities

Part 2.3: For node.js / front-end dev

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

For node.js / front-end dev

Dependency Managers



<https://www.npmjs.com/>

- main central repository for js
- package.json - example: <https://github.com/angular/angular-cli/blob/master/package.json>



<https://yarnpkg.com/>

- npm + caching (limits downloads) + parallelization

For node.js / front-end dev

Tasks automation



grunt <https://gruntjs.com/>

- task automation: minification, linting, testing, ...

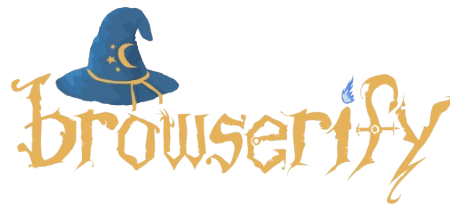


gulp <https://gulpjs.com/>

- grunt + faster (ram vs i/o) + big ecosystem (plugins)

For node.js / front-end dev

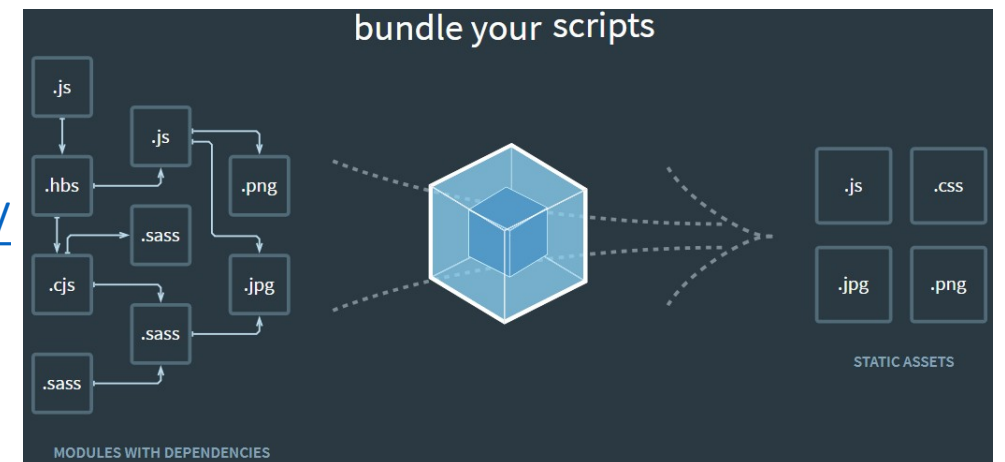
bundling of code and dependencies so that the client downloads only one js file and one css file

 browserify <http://browserify.org/>

- package the code and the dependencies ([require\(\)](#) fonction of Node.js)
- only one js

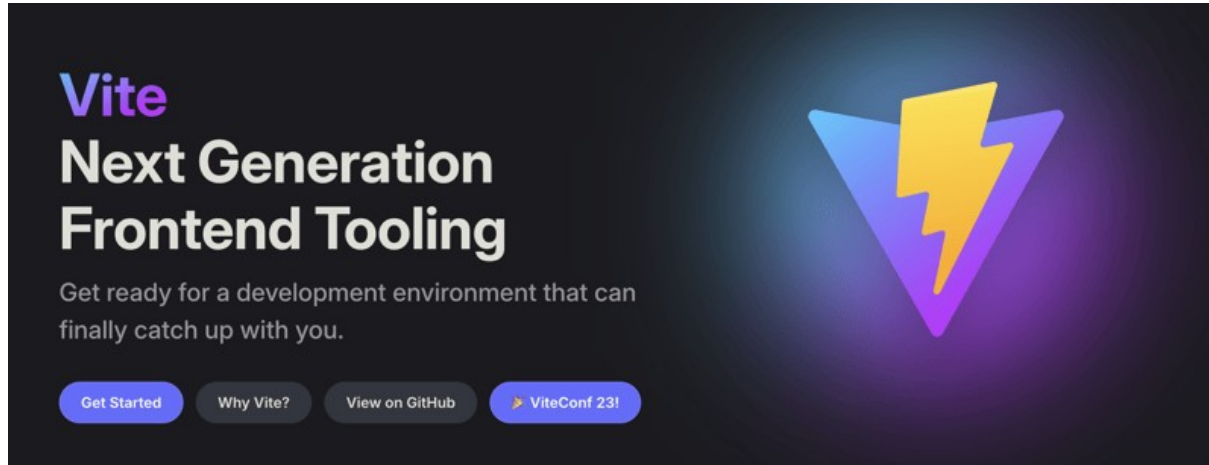
 **webpack** webpack <https://webpack.js.org/>

- The solution to recommend today

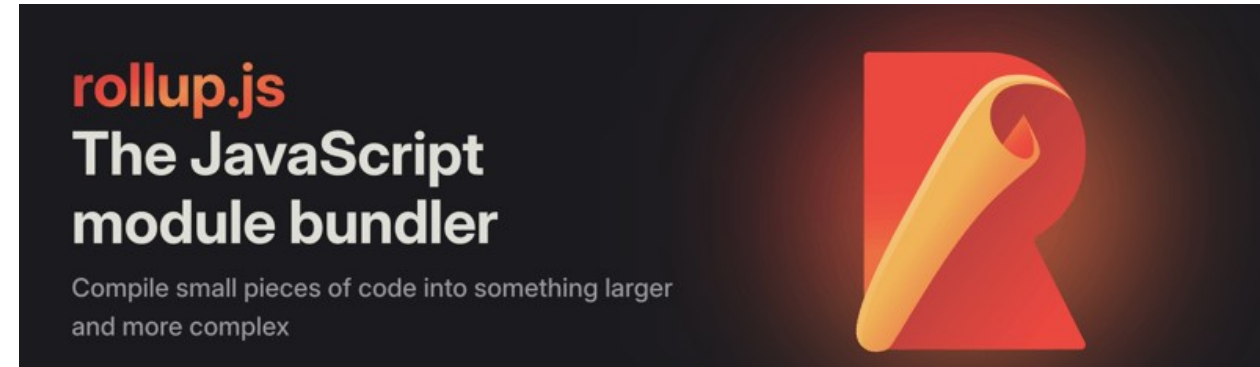


webpack was recommended in 2022.
In 2024 there are these alternatives:

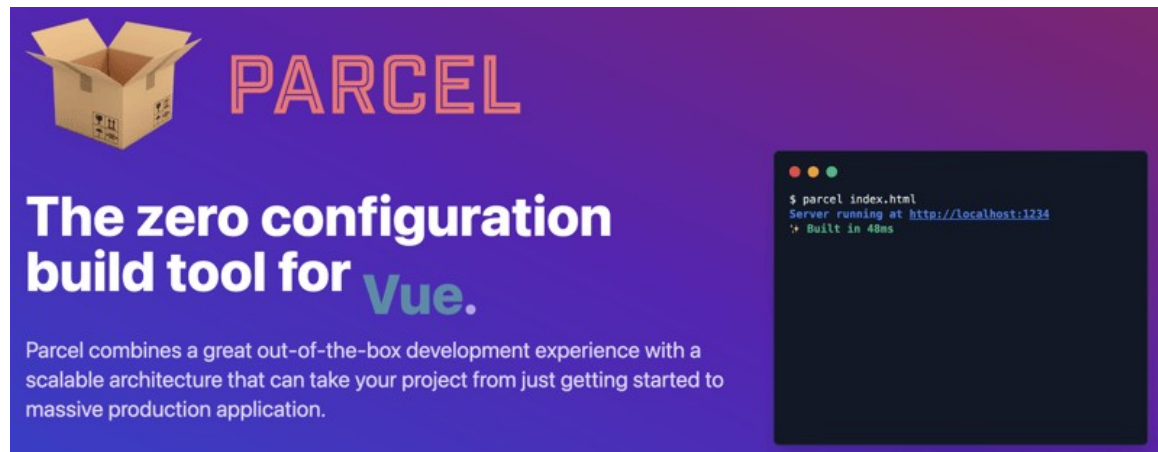
For node.js / front-end dev



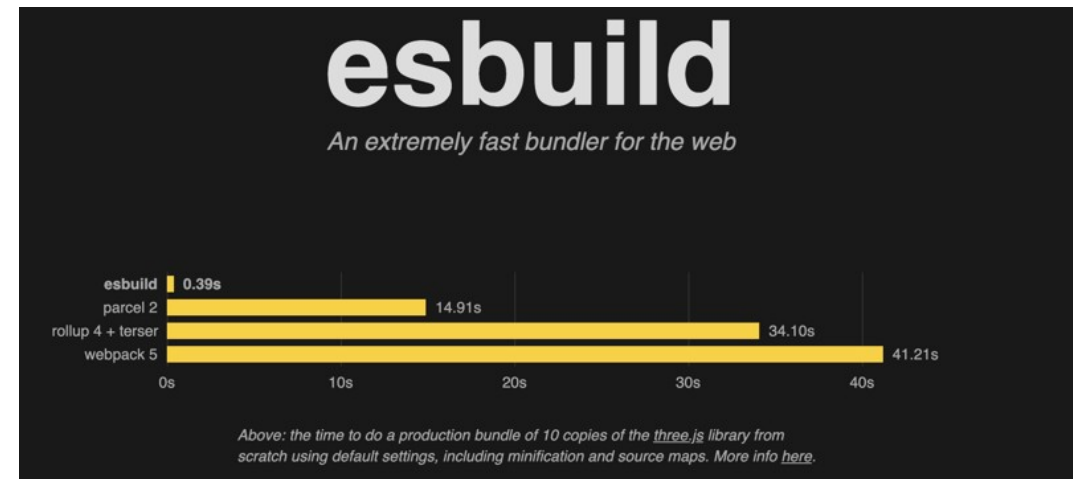
<https://vitejs.dev/>



<https://rollupjs.org/>



<https://parceljs.org/>



<https://esbuild.github.io/>

Technological foundations of software development

Automate build production

Part 1: Build automation utilities

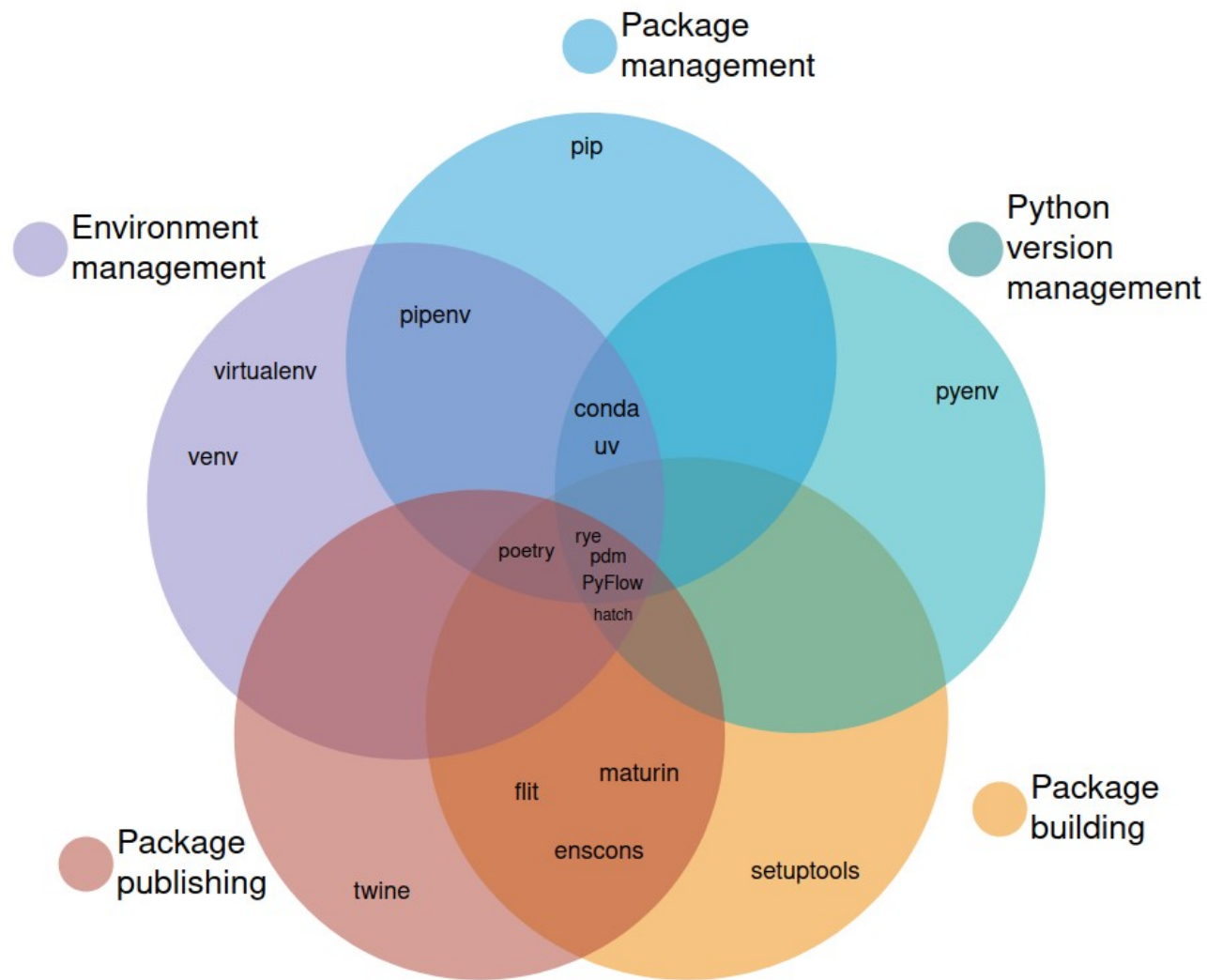
Part 2.4: For python

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

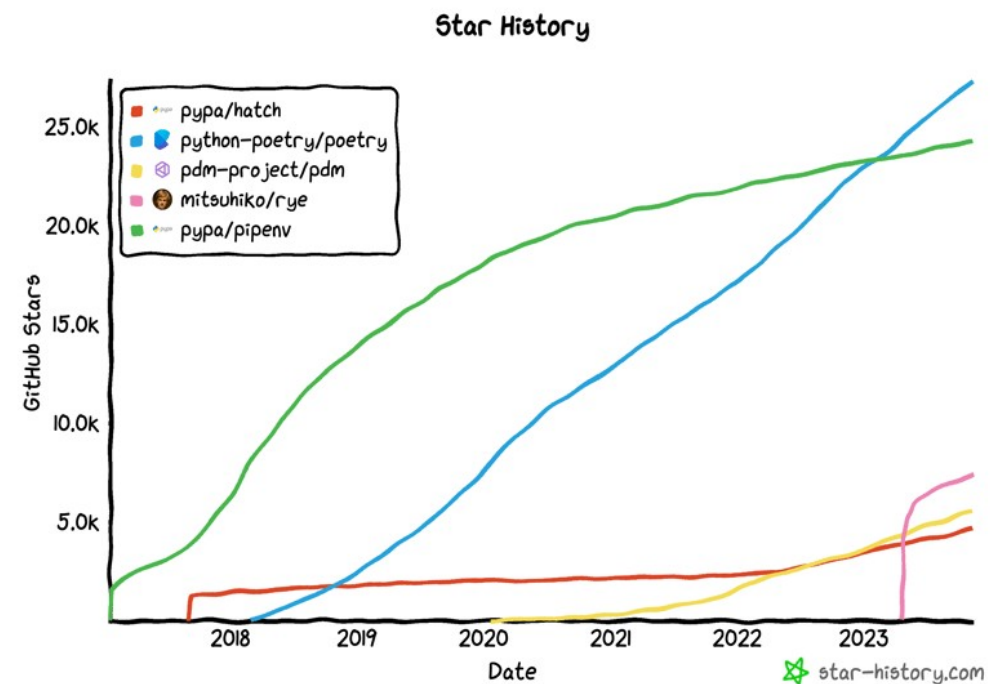
Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>



https://alpopkes.com/posts/python/packaging_tools/

	Flit	Poetry	PDM	Hatch	Rye	uv
Does the tool manage dependencies?	✗	✓	✓	✗	✓	✓
Does it resolve/lock dependencies?	✗	✓	✓	✗	✓	✓
Is there a clean build/publish flow?	✓	✓	✓	✓	✓	✗
Does it allow to use plugins?	✗	✓	✓	✓	✗	✗
Does it support PEP 660 (editable installs)?	✓	✓	✓	✓	✓	✓
Does it support PEP 621 (project metadata)?	✓	✗	✓	✓	✓	✓



<https://dev.to/adamghill/python-package-manager-comparison-1g98> (2023)



Poetry

Example: Poetry

Poetry

dependency management, linting, autoformatting, testing, and publishing in python

<https://python-poetry.org/>

MIT License - Open source: <https://github.com/python-poetry/poetry>

- Managing different environments
- Installing python packages
- Environment reproducibility
- Packaging and publishing python packages

```
$ poetry init / poetry install  
$ poetry add "package==version"  
$ poetry update  
$ poetry run  
$ poetry shell - exit
```

See <https://medium.com/edge-analytics/python-best-practices-2934de825fd2>



Example: Hatch

Hatch

dependency management, linting, autoformatting, testing, and publishing in python

<https://hatch.pypa.io/latest/>

MIT License - Open source: <https://github.com/pypa/hatch>

- Standardized [build system](#) with reproducible builds by default
- Robust [environment management](#) with support for custom scripts and UV
- Configurable [Python distribution management](#)
- [Test execution](#) with known best practices
- [Static analysis](#) with sane defaults
- Built-in Python [script runner](#)
- Easy [publishing](#) to PyPI or other indices
- [Version](#) management
- Best practice [project generation](#)
- Responsive [CLI](#), ~2-3x [faster](#) than equivalent tools

To read for MCQ test

- To read (necessary for questions in the MCQ)
 - The pyproject.toml guide
<https://packaging.python.org/en/latest/guides/writing-pyproject-toml/>
 - “An unbiased evaluation of environment management and packaging tools”
https://alpopkes.com/posts/python/packaging_tools/

Technological foundations of software development

Automate build production

ICM – Computer Science Major – Course unit on Technological foundations of computer science

M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development

Maxime Lefrançois <https://maxime-lefrancois.info>

online: <https://ci.mines-stetienne.fr/cps2/course/tfsd/>

... Your turn

Complete the TODO section:

https://ci.mines-stetienne.fr/cps2/course/tfsd/course-4.html#_todos