Practical Software Engineering 2.

Build systems

Máté Cserép
ELTE IK
2019.

Build systems

Compiling Java programs

```
javac -d dist
  src\thesisGenerator\*.java
  src\thesisGenerator\model\*.java
  src\thesisGenerator\view\*.java


cd dist
jar -cfe thesis-generator.jar
  thesisGenerator.ThesisGenerator
  thesisGenerator\*.class
  thesisGenerator\model\*.class
  thesisGenerator\view\*.class


java -jar thesis-generator.jar
```

Build systems

Requirements towards build systems

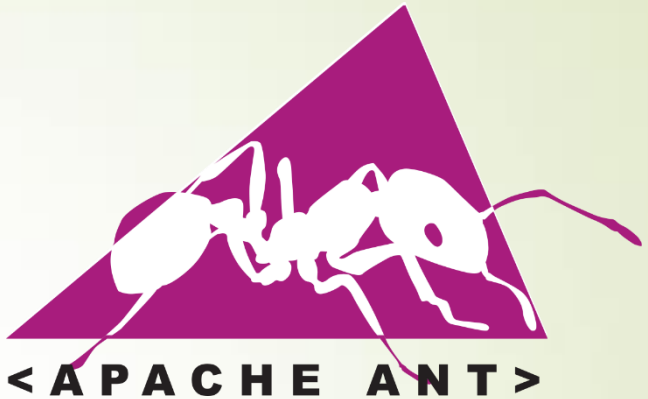
- Compiling the code
  - Manage dependencies of compilation targets
- Packaging the binaries
  - Support multiple release options
- Perform automatized tests
- Deploying the binaries to the test server
- Copying the code from one location to another

- Management of package repositories

Build systems - Ant

Features

- Imperative approach
- Typically used for Java projects
- XML-based build file
  - Named *build.xml* by default



- https://ant.apache.org/
- https://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

Build systems - Ant

Build file (build.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="projname"
         default="deftarget"
         basedir=".">
  ...
</project>
```

Build systems - Ant

Define a target

```
<project ...>
  <property name="jarname"
            value="filename.jar" />

  <target name="compile" depends="prepare">
    ...
  </target>
</project>
```

- Command line:
  `ant` **`compile`**

Build systems - Ant

Directory creation

```
<project ...>
  <target name="prepare">
    <mkdir dir="classes"/>
  </target>
</project>
```

- Command line:
  `ant` **`prepare`**

Build systems - Ant

Target dependencies

```
<project ...>
  <target name="compile" depends="prepare">
    <javac srcdir="src" destdir="classes"/>
  </target>
</project>
```

- Command line:
  `ant` **`compile`**

Build systems - Ant

Cleanup target

```
<project ...>
  <target name="clean">
    <delete>
      <fileset dir="classes" includes="*"/>
    </delete>
    <delete dir="classes"/>
  </target>
</project>
```

- Command line:
  `ant` **`clean`**

Build systems - Ant

Cleanup target

```
<project ...>
  <target name="clean" failonerror="false">
    <delete dir="classes"/>
  </target>
</project>
```

- Command line:
  `ant` **`clean`**

Build systems - Ant

Packaging

```
<project ...>
  <target name="jar" depends="compile">
    <jar destfile="${jarname}">
      <fileset dir="classes">
        <include name="*.class"/>
      </fileset>
      <manifest>
        <attribute name="Main-Class" value="Main"/>
      </manifest>
    </jar>
  </target>
</project>
```

Build systems - Ant

Target: complex example

```
<target name="compile" depends="prepare,init">
  <javac destdir="build/classes" debug="on">
    <src path="src1/java"/>
    <src path="src2/java"/>
    <include name="**/*.java"/>
    <exclude name="com/comp/xyz/applet/*.java"/>
    <classpath>
      <fileset dir="lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </javac>
</target>
```

Build systems - Ant

Deploying (file operations)

```
<target name="install" depends="jar">
  <mkdir dir="build/war/WEB-INF/lib"/>
  <copy todir="build/war/WEB-INF/lib">
    <fileset dir="lib">
      <include name="*.jar"/>
      <exclude name="servlet-api.jar"/>
      <exclude name="catalina-ant.jar"/>
      <exclude name="el-api.jar"/>
    </fileset>
  </copy>
</target>
```

- Command line:
  `ant` **`install`**

Build systems - Ant

JVM launch

```
<target name="run">
  <java classname="com.comp.foo.TestClient"
        jvmargs="-Xdebug server=y,suspend=n">
    <classpath>
      <fileset dir="lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </java>
</target>
```

- Command line:
  `ant` **`run`**

Build systems - Ant

Generating API documentation

```
<target name="doc">
  <tstamp>
    <format property="timestamp" pattern="d.M.yyyy"
            locale="en"/>
  </tstamp>
  <mkdir dir="doc"/>
  <javadoc sourcepath="src" destdir="doc"
           windowtitle="Project documentation">
    <header>Very Important Project</header>
    <footer>Javadocs compiled ${timestamp}></footer>
    <fileset dir="src/" includes="**/*.java" />
  </javadoc>
</target>
```

Build systems - Ant

Invocation of target

- Automatic invocation of dependencies with manual invocation of an other target (e.g. with different build settings):

```
<antcall target="targetname"/>
```

Build systems - Ant

NetBeans

- By default Netbeans uses Ant as a build system.
  - *build.xml* is located in the project root
  - references *nbproject/build-impl.xml*, which is generated by Netbeans and shouldn't be modified
  - Targets as hooks can be definied in *build.xml*, which will be called by Netbeans's build process automatically:
    - *-pre-init*, *-post-init*, *-pre-compile*, *-post-compile*, *-pre-jar*, *-post-jar*, *-post-clean*, etc.

Build systems - Maven

Features

- Software project management tool
- Building project, running tests, managing dependencies, documentation
- Packages: automatic download of dependencies
- Declarative specification of the build process
- Fix, predefined directory structure, conventions
- Typically Java, but it can handle other language via plugins
- XML-based build file
  - Named *pom.xml* by default

- http://maven.apache.org/
- https://www.baeldung.com/maven

Build systems - Maven

Project

- Project Object Model (POM)
- Project uniquely identified by project's group, artifact Id, version (GAV)
- Project can divided to multiple modules that can be handled independently

Build systems - Maven

Project Object Model (pom.xml)

- Specification of the project's important information:
  - groupId, artifactId, version
  - how it is build
  - what is the result of the build
  - testcases
  - dependencies

Build systems - Maven

Project Object Model (pom.xml)

```
<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>com.mycompany.software</groupId>
  <artifactId>app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  ...
</project>
```

Build systems - Maven

Directory structure

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   |-- java
    |   |   `-- com
    |   |       `-- mycompany
    |   |           `-- app
    |   |               `-- App.java
    |   `-- resources
    |       `-- META-INF
    |           |-- application.properties
    `-- test
        |-- java
        |   `-- com
        |       `-- mycompany
        |           `-- app
        |               `-- AppTest.java
        `-- resources
            `-- test.properties
```

Build systems - Maven

Directory structure override

```
<project>
   ...
   <build>
     <directory>target</directory>
     <outputDirectory>classes</outputDirectory>
     <finalName>${project.artifactId}-${project.version}</finalName>
     <testOutputDirectory>test-classes</testOutputDirectory>
     <sourceDirectory>src/main/java</sourceDirectory>
     <scriptSourceDirectory>src/main/scripts
     </scriptSourceDirectory>
     <testSourceDirectory>/src/test/java</testSourceDirectory>
     ...
   </build>
</project>
```

Build systems - Maven

Build phases

- **validate:** validate the project is correct and all necessary information is available
- **compile:** compile the source code of the project
- **test:** test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package:** take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test:** process and deploy the package if necessary into an environment where integration tests can be run

Build systems - Maven

Build phases

- **verify:** run any checks to verify the package is valid and meets quality criteria
- **install:** install the package into the local repository, for use as a dependency in other projects locally
- **deploy:** done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.
- Further details: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
- Command line: `mvn clean install`

Build systems - Maven

Goals

- Compilation phases consist of goals
- The goal is a task that is related to the project's compilation or management
- Multiple goals can be passed to the Maven in command
- The order of these goals depends on the phase's binding
- Goals can be defined (in the *pom.xml*)

Build systems - Maven

Repositories

- Central: http://repo.maven.apache.org
- Artifact repositories
- Local: own repository on the developer's computer:
  - *~/.m2/repository*

Build systems - Maven

Repositories

- Maven downloads the dependent artifacts and plugins at the first build, stored in the local repository
- Network traffic and build time are significantly increased
- Incremental changes
- Maven can be configured to use the specified repo or mirror: *~/.m2/settings.xml*

Build systems - Maven

Result of build process

- *target* directory is created during compilation which stores the new files that generated at compilation time
  - output, e.g. *my-app-1.0-SNAPSHOT.jar*
  - *classes* directory – class files that created during compilation but not test classes
  - *test-classes*: classes created from test sources
  - *maven-archiver-pom.properties* file that defines the project's GAV
  - *surefire-reports:* reports of the tests

Build systems - Maven

Project hierarchies

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>parent-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <!-- subprojects -->
  <modules>
    <module>first-child-app</module>
    <module>second-child-app</module>
  </modules>
</project>
```

Build systems - Maven

Project hierarchies

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>parent-app</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <groupId>com.mycompany.app</groupId>
  <artifactId>first-child-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  ...
</project>
```

Build systems - Maven

Plugins

- Maven’s functionality limited to the basic
- Many different plugins are available
  - e.g. C++, LaTeX, ant build, javadoc, etc.
- One can write own plugin

Build systems - Maven

Plugin example: Javadoc

```
<project ...>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.8.1</version>
        <configuration>
          ...
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

- Documentation generation: `mvn javadoc:javadoc` or `mvn:site`

Build systems - Maven

Plugin example: Javadoc

- Generate HTML page for Javadoc:
  - Perform the javadoc:javadoc goal
  - mvn javadoc:javadoc
  - Build a site

Build systems - Maven

Dependencies

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- GAV uniquely specifies the required artifact
- scope defines how we use the the dependency

Build systems - Maven

Dependencies' scope

- The most important scopes:
  - **compile:** This is the default if unspecified. Dependencies that required by the compilation
  - **runtime:** Dependency required at runtime, but not required at compilation time.
  - **test:** Dependency is not required in production but it is required for the compilation and execution of testcases.

Build systems - Maven

Netbeans

- Netbeans also supports Maven-based projects.