maven

zostavovanie projektov
celkom ľahko

Robert.Novotny@upjs.sk

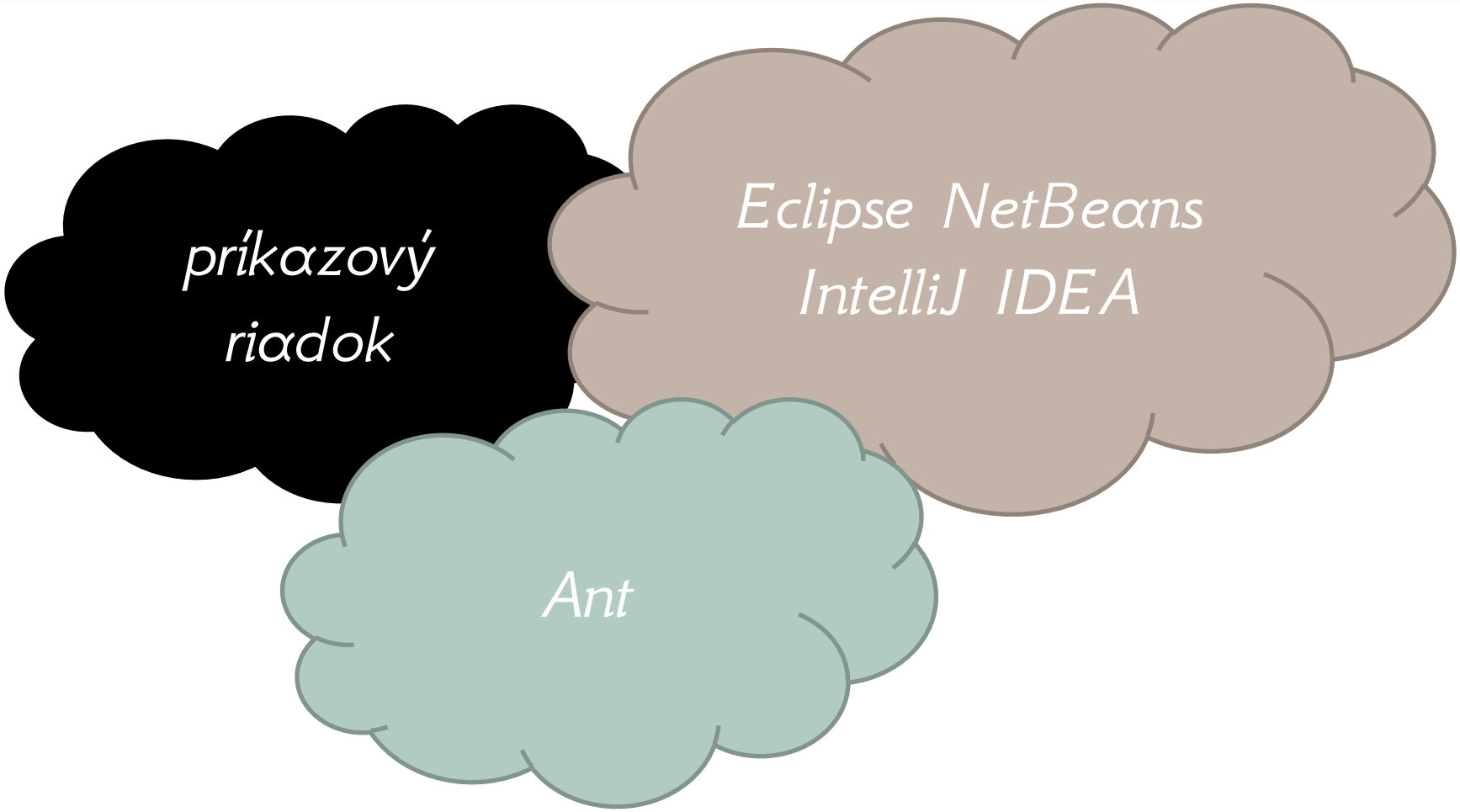
Potrebujem zostaviť tento ...

„Do jedného megaJARu potrebujem napchať logovanie SLF4j, Apache HTTP Client, ďalší modul pre spracovanie súborov. To všetko podpísať a zavesiť na web pod menom applet.jar"

Potrebujem zostaviť tento ...

„Potrebujem vyvíjať modul pre perzistentnú vrstvu a nezávisle od toho modul pre webové rozhranie. Na záver potrebujem všetko zabaliť do WARu a nasadiť na Glassfish"

Iné riešenia?



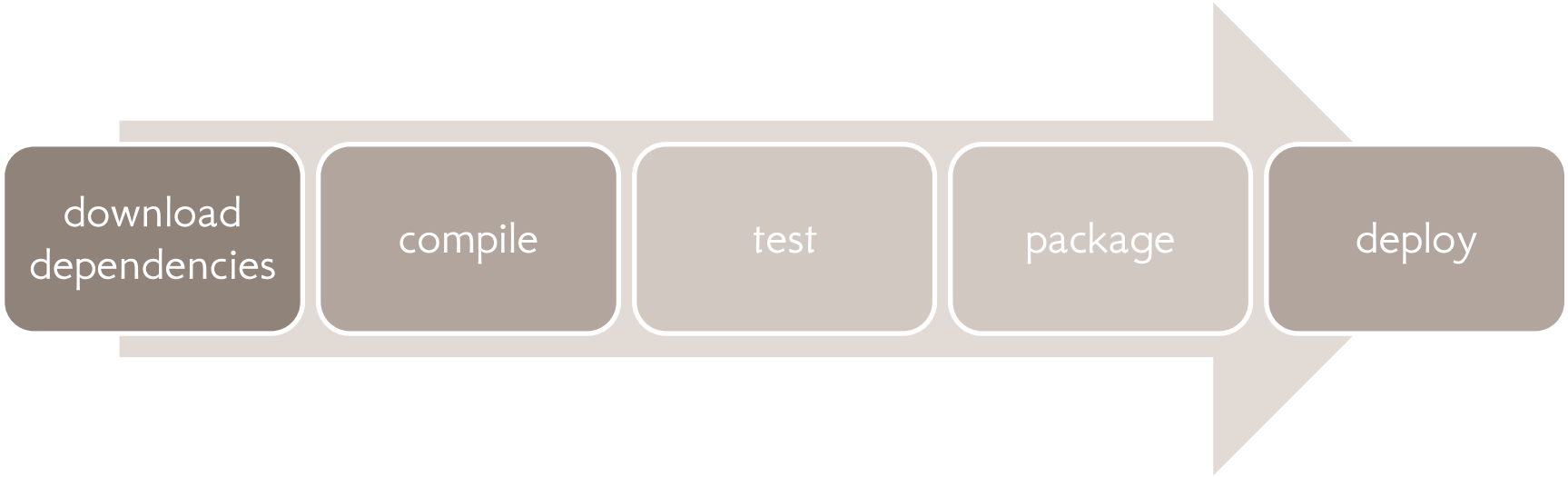
Maven

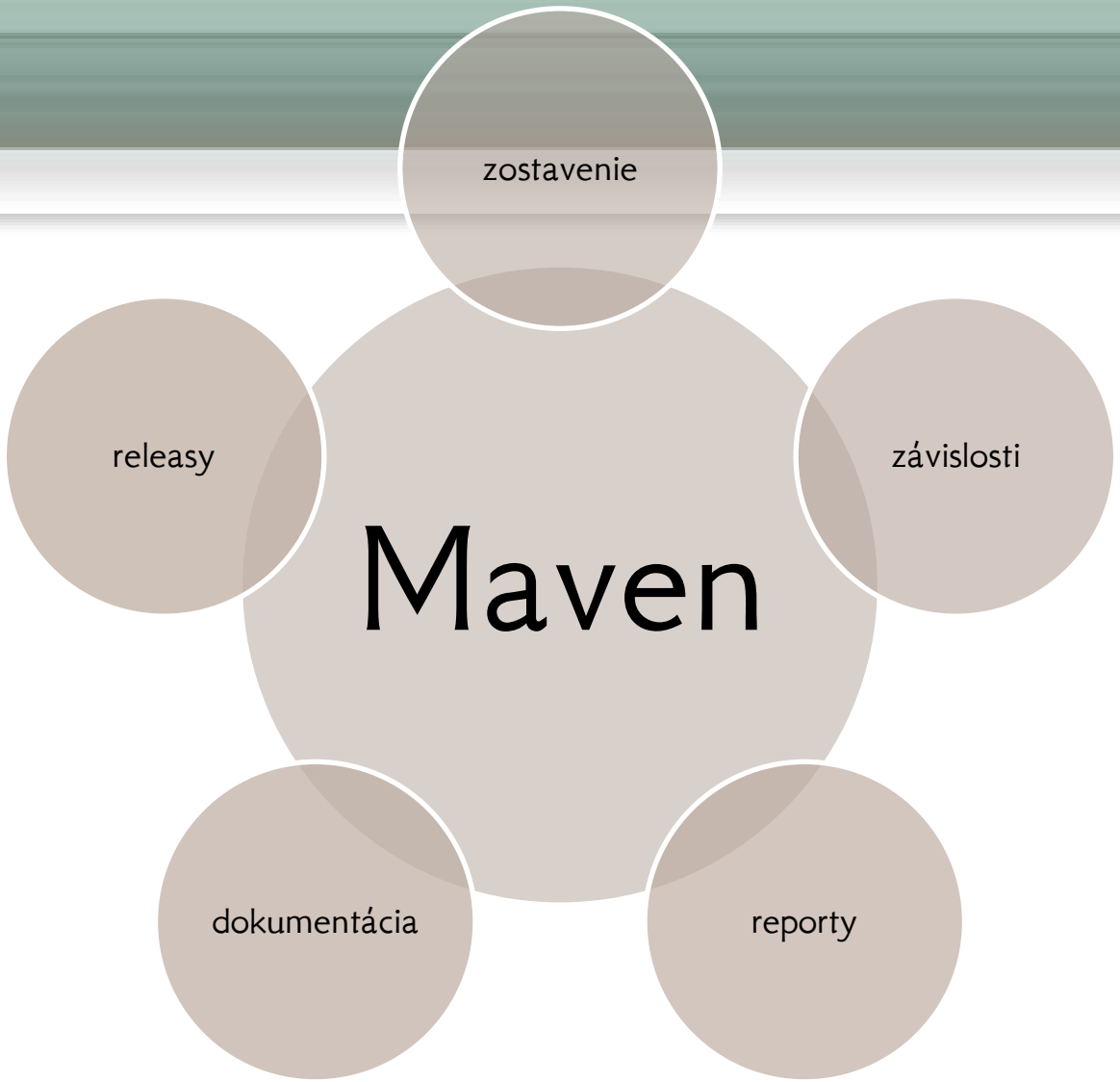
„znalec"

jidiš < hebr. mevin
(ten, kto chápe, rozumie)

Apache Maven

- build management tool
- konzistentné a jednotné zostavenie projektu
- postavený na konvenciách




Maven
zostavenie
závislosti
reporty
dokumentácia
releasy

Ako začať?

- http://maven.apache.org
- momentálne verzia 3.0
- rozbaliť
  - napr. do C:/Java/maven
  - nastaviť premennú prostredia M2_HOME=C:/Java/maven
  - nastaviť PATH: C:/java/maven/bin

Ako začať?

- Prvé spustenie

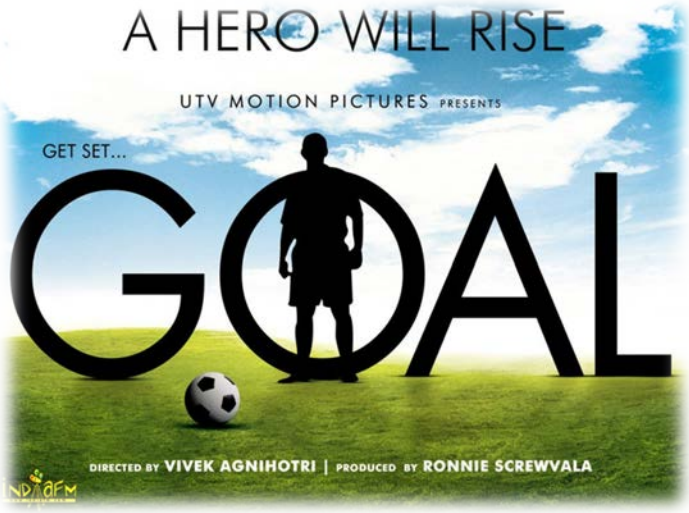
mvn

```
c:\Projects>mvn
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.094s
[INFO] Finished at: Mon Mar 19 11:49:57 CET 2012
[INFO] Final Memory: 6M/120M
[INFO] ------------------------------------------------------------------------
[ERROR] No goals have been specified for this build. You must specify a valid lifecycle phase or a goal in
>:<plugin-artifact-id>[:<plugin-version>]:<goal>. Available lifecycle phases are: validate, initialize, gen
rocess-resources, compile, process-classes, generate-test-sources, process-test-sources, generate-test-reso
test-classes, test, prepare-package, package, pre-integration-test, integration-test, post-integration-test
 site-deploy, pre-clean, clean, post-clean. -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/NoGoalSpecifiedException
```

Ako začať?

- Maven potrebuje vedieť, **čo** má urobiť
- Chce vedieť **goal**

Prvý cieľ: nový projekt!



Založenie projektu pre enterovačov

mvn archetype:generate

- spustiť z adresára s projektami
  - napr. v C:/Projects
- dve minúty pauzy na WC
- Enter, Enter
- groupId: sk.upjs.ics
- artifactId: hellomaven
- Enter, Enter, y
- voilá, nový projekt v C:/Projects/hellomaven!

http://rodcom1000.deviantart.com/art/Maven-126212535

Maven je spúšťač plug-inov

```
mvn archetype:generate
```

- v Mavene je takmer všetko plug-inom
- počas pauzy na WC sa ťahali plug-iny z **centrálneho úložiska**
  - treba konektivitu
  - plug-iny sa kešujú v lokálnom úložisku
- spustili sme goal **generate** z plug-inu **archetype**

Archetypy

```
mvn archetype:generate
```

- **archetyp = kostra** aplikácie s predpripravenými súbormi
- v centrálnom úložisku: 538 možností
- **prvý Enter:** výber default archetypu
  - maven-archetype-quickstart
- **druhý Enter:** výber default verzie archetypu

Co śie stao?

- nová adresárová štruktúra + súbory!
- vďaka ti, ó, archetyp!

```
C:.
│   pom.xml
└───src
    ├───main
    │   └───java
    │       └───sk
    │           └───upjs
    │               App.java
    │
    └───test
        └───java
            └───sk
                └───upjs
                    AppTest.java
```

Konvencie v Mavene

- **src/main/java**: zdrojáky *.java
- **src/test/java:** zdrojáky *.java pre unit testy
- **src/main/resource:** konfiguráky, META-INF...

Nebojujte s konvenciami!

```
C:.
│   pom.xml
└───src
    ├───main
    │   └───java
    │       └───sk
    │           └───upjs
    │               App.java
    │
    └───test
        └───java
            └───sk
                └───upjs
                    AppTest.java
```

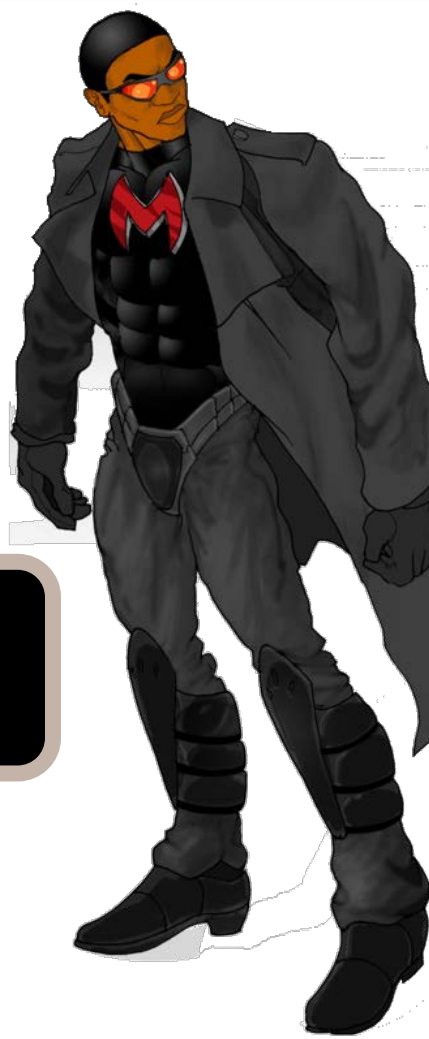
Vytvorenie JARu

```
mvn package
```

- z adresára v ktorom je projekt
  - C:/Projects/hellomaven
- výsledok:

```
mavenhello-1.0-SNAPSHOT.jar
```

- objaví sa v adresári **target**



http://rodcom1000.deviantart.com/art/Maven-126212535

Vytvorenie JARu

```
c:\Projects\hellomaven>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building mavenhello 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ mavenhello ---
[debug] execute contextualize
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory c:\Projects\hellomaven\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ mavenhello ---
[INFO] Compiling 1 source file to c:\Projects\hellomaven\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ mavenhello ---
[debug] execute contextualize
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory c:\Projects\hellomaven\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ mavenhello ---
[INFO] Compiling 1 source file to c:\Projects\hellomaven\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.10:test (default-test) @ mavenhello ---
[INFO] Surefire report directory: c:\Projects\hellomaven\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running sk.upjs.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.01 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ mavenhello ---
[INFO] Building jar: c:\Projects\hellomaven\target\mavenhello-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.031s
[INFO] Finished at: Mon Mar 19 13:12:59 CET 2012
[INFO] Final Memory: 17M/152M
[INFO] ------------------------------------------------------------------------
```

Čo sa stalo?

- skompilovali sa zdroják y
- zbehli unit testy
- vyrobil sa JAR
- JAR = hmatateľný výsledok

Vykonal sa **build lifecycle**

Spustenie aplikácie

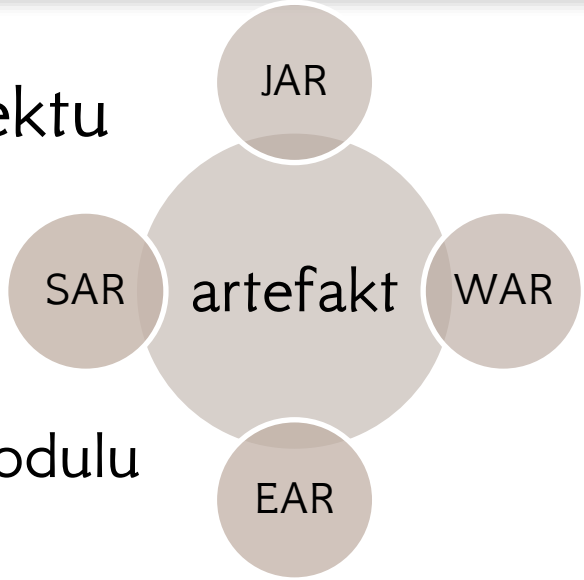
```
java -cp target/mavenhello-1.0-SNAPSHOT.jar sk.upjs.App
```

- z adresára v ktorom je projekt
  - C:/Projects/hellomaven
- výsledok:

```
Hello World!
```

Artefakty a identifikátory

- **artefakt:** hmatateľný výstup projektu
- jednoznačne identifikovaný
  - **groupId:** názov firmy
  - **názov artefaktu:** názov projektu/modulu
  - **verzia**



com.upjs.ics:hellomaven:1.0.0-SNAPSHOT

Artefakt nie je archetyp!

Archetype [archetype]

- kostra aplikácie
- predpripravená štruktúra adresárov
- použitá raz, pri vytváraní projektu

Artefakt [artifact]

- hmatateľný výstup
- obvykle JAR / WAR...
- z projektu vypadne 1 artefakt

pom.xml – popisovač projektu

- jednoznačne identifikuje projekt
- + identifikátor
- + metadáta
- + závislosti
- + druh artefaktu
- + spôsob jeho buildovania

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

  <groupId>sk.upjs</groupId>
  <artifactId>mavenhello</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

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

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

  <dependencies>
        <!-- -- >
  </dependencies>
</project>
```

Popisovač artefaktu

Krajší názov projektu + webové sídlo

Kódovanie zdrojákov

Závislosti

Závislosti

- projekt má často závislosti
- artefakt závidí na iných artefaktoch
- logovanie cez slf4j? dodajme do pom.xml

```
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.6.4</version>
</dependency>
```

*„Žiaden projekt nie je **ostrovom** samým pre seba…“*

Závislosti

```
package sk.upjs;

import com.slf4j.*;
/**
 * Hello world!
 *
 */
public class App {
  public final Logger logger = LoggerFactory.getLogger(App.class);
  public static void main(String[] args ) {
    logger.info( "Hello World!" );
  }
}
```

Vytvorenie JARu

mvn package

- stiahnu sa závislosti
- projekt sa skompiluje
- vytvorí sa JAR

http://rodcom1000.deviantart.com/art/Maven-126212535

The Central Repository

- funguje filozofia balíčkovacích správcov z Linuxu
- Maven Central: úložisko artefaktov projektov na webe
  - prítomný takmer každý známy projekt
- Maven preberá závislosti práve odtiaľ
- možnosť vyhľadávať na:

http://search.maven.org

Maven Local

- šetríme traffic: lokálna cache v profile

C:/Users/novotnyr/.m2

- závislosti sa hľadajú najprv v lokálnej cache
- ak sa nenájdu, ide sa do Centralu
  - odtiaľ sa stiahnu do lokálnej cache

Inštalácia plug-inu

mvn install

- hotový projekt bude možno používaný inými modulmi
- treba ho zverejniť
- **mvn install** ho nainštaluje do lokálnej cache

Distribúcia plug-inu

mvn deploy

- publikuje artefakt do projektového úložiska
  - do Maven Central (máte práva? ☺)
  - alebo do firemného úložiska
  - nastavenia v dokumentácii Mavenu

Plug-iny

„Potrebujem otestovať webový projekt v Jetty!"

- plug-iny rozširujú funkcionalitu = dodávajú nové goaly
- deklarované v pom.xml
- **konvencia**: veci súvisiace s webom v src/main/webapp

Jetty plug-in

plug-in je tiež artefakt

```
<project ... >
  <build>
    <plugins>
      <plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>8.1.2.v20120308</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Aby to naozaj fungovalo..

```
<project ... >
  ...
  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>6.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  ...
</project>
```

závislosť pre servlety

o scopes neskôr

závislosť, ktorá sa nepribalí medzi knižnice, lebo ju dodá napr. servletový kontajner

Ešte jeden servlet...

```
package sk.upjs.hellomaven;

import java.io.*; import javax.servlet.*; import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns="/Hello")
public class HelloServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req,
      HttpServletResponse resp) throws ServletException, IOException
  {
    PrintWriter out = resp.getWriter();
    out.println("Hello world!");
  }
}
```

Spustenie projektu

mvn jetty:run

- stiahnu sa závislosti
- projekt sa skompiluje
- vytvorí sa JAR
- spustí sa servletový kontajner Jetty
- deployne doň aplikácia
- dostupná na http://localhost:8080

chrústanie závislostí

http://rodcom1000.deviantart.com/art/Maven-126212535

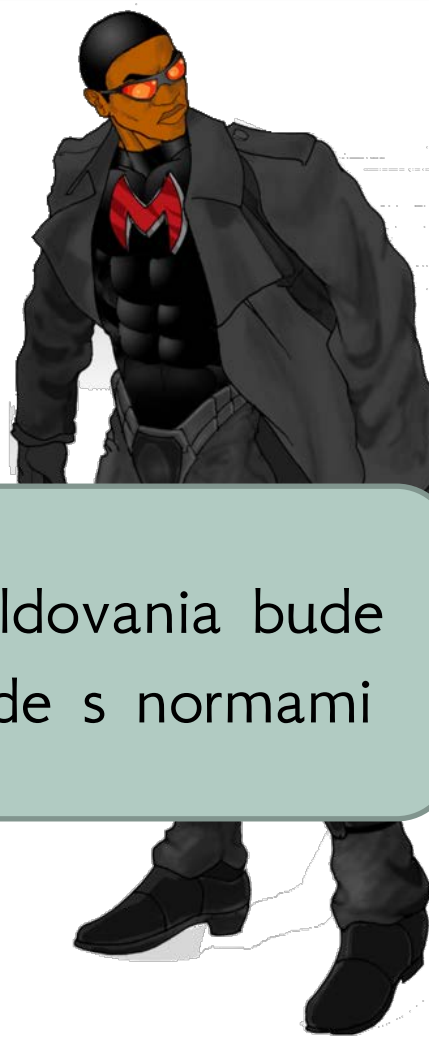
Vytvorenie WAR

```
mvn jetty:run
```

- štandardne produkujeme JARy
- zmena na WAR = jediná úprava

```
<project .... >
  ...
  <packaging>war</packaging>
  ...
</project>
```

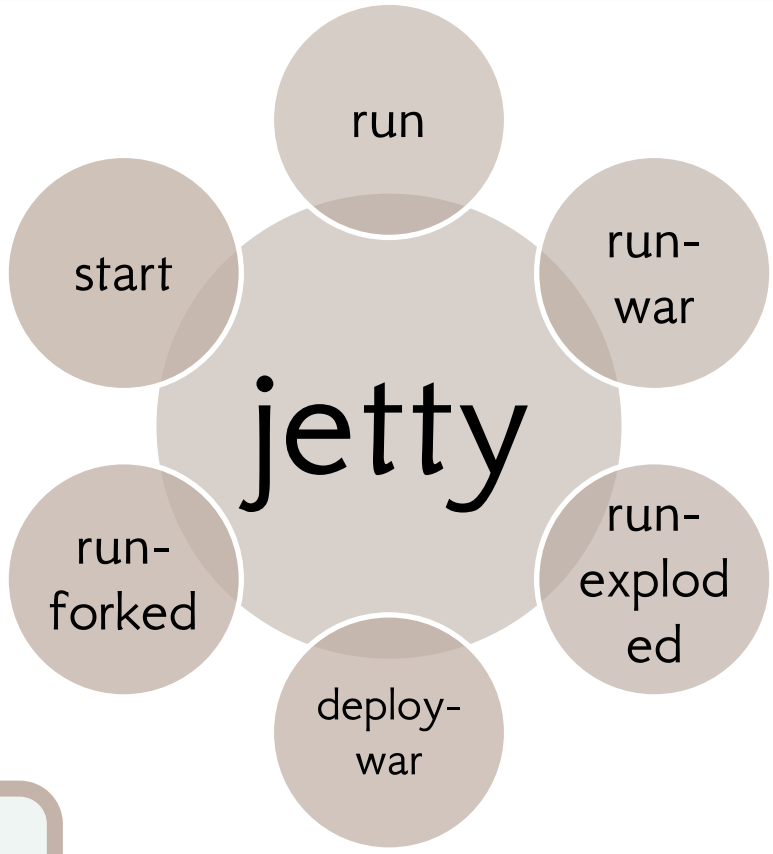
výstupom buildovania bude WAR v súlade s normami



PLUG-INY

Plug-iny, mojo a goaly

- každý plug-in ponúka viacero goalov
- 1 goal = riešenie 1 konkrétnej úlohy
- goaly možno volať z príkazového riadku




```
mvn jetty:run
```

Spustenie ľubovoľného plug-inu

```
mvn org.mortbay.jetty:jetty-maven-plugin:
8.1.2.v20120308:run
```

groupId:artifactId:version:goal

- plug-in musí byť dostupný v úložisku
- alternatívne konvencia:

```
mvn jetty:run
```

Spustenie ľubovoľného plug-inu

```
mvn org.apache.maven.plugins:maven-dependency-plugin:2.1:copy-dependencies
```

groupId:artifactId:version:goal

- plug-in musí byť dostupný v úložisku
  - Maven Central
  - či iné repositories
- spustí sa s default konfiguráciou

Spustenie ľubovoľného plug-inu

```
mvn org.apache.maven.plugins:maven-dependency-plugin:2.1:copy-dependencies
```

- možno vynechať groupId, ak je:
  - org.apache.maven.plugins
  - org.codehaus.mojo
- a možno vynechať verziu

```
mvn dependency:copy-dependencies
```

Spustenie ľubovoľného plug-inu

- ak plug-in zavedieme do pom.xml...

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId>
  <version>8.1.2.v20120308</version>
</plugin>
```

- môžeme jeho goaly tiež volať konvenciou

```
mvn jetty:run
```

Čo znamená 'mvn package'?

- Maven potrebuje uviesť buď cieľ alebo fázu:

goal

jetty:run

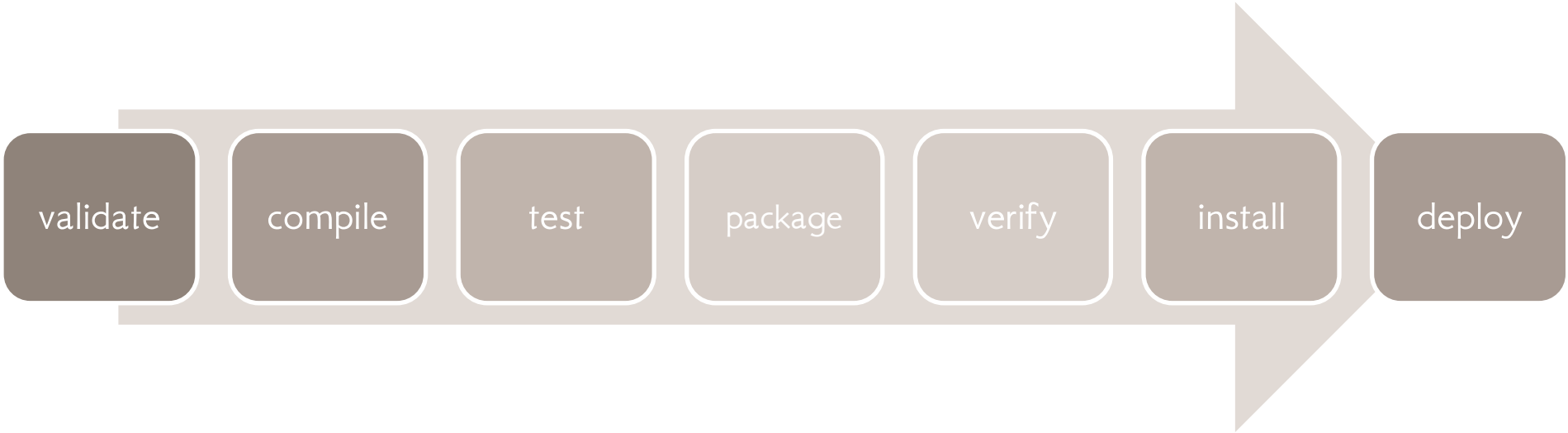
alebo

build lifecycle phase

package

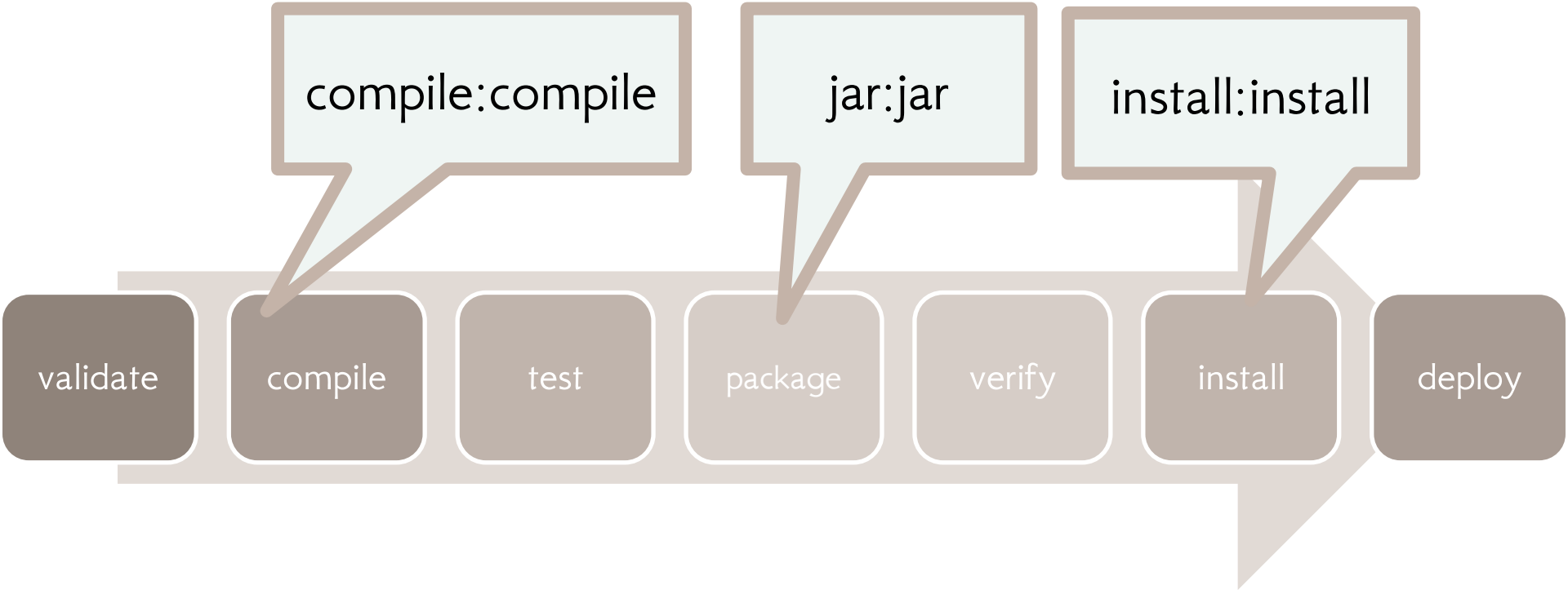
Build lifecycle

- proces „z ničoho až k zverejneniu artefaktu v úložisku"
- má viacero fáz
- fázu možno zavolať z príkazového riadka



Build lifecycle: fázy

- na každú fázu sa viaže jeden či viac goalov
- detaily závisia od typu projektu a konfigurácie



Naviazanie plug-inu na fázu

```
<plugin>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.1 </version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

vo fáze „package" zavolaj goal „dependency:copy-dependencies"

Naviazanie plug-inu na fázu

- odteraz stačí zavolať

```
mvn package
```

- prebehnú všetky fázy až po package…
- …plus goal dependency:copy-dependencies
  - nakopírujú sa závislosti do **target/dependencies**

Naviažme goal „assembly:single“ na fázu „package“

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.3</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals><goal>single</goal></goals>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

vo fáze „package“ zavolaj goal „assembly:single“

Vyrobil som überJAR!

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.3</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals><goal>single</goal></goals>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Dodatočná konfigurácia plug-inu

Konzistentnosť buildov

- buildovanie je konzistentné bez ohľadu na typ a rozsah projektu
- web projekt zbuildujem rovnako ako Swing

mvn package

dependency managemenet

SPRÁVA ZÁVISLOSTÍ

Správa závislosti

- uvádzané do <dependency>
- podpora tranzitívnych závislosti

projekt závisí na Apache HTTP Client

Apache HTTP Client závisí na
- Jakarta Commons Codec
- Jakarta Commons Logging

Tranzitívne závislosti

```
<project … >
  …
  <dependencies>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.1.3</version>
    </dependency>
  </dependencies>
    …
</project>
```

stačí uviesť koreňovú závislosť

mvn dependency:tree

- automaticky sa zavedú závislosti závislostí

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building mavenhello 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ mavenhello ---
[INFO] sk.upjs:mavenhello:jar:1.0-SNAPSHOT
[INFO] +- junit:junit:jar:3.8.1:test
[INFO] +- org.slf4j:slf4j-api:jar:1.6.4:compile
[INFO] \- org.apache.httpcomponents:httpclient:jar:4.1.3:compile
[INFO]    +- org.apache.httpcomponents:httpcore:jar:4.1.4:compile
[INFO]    +- commons-logging:commons-logging:jar:1.1.1:compile
[INFO]    \- commons-codec:commons-codec:jar:1.4:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.055s
[INFO] Finished at: Wed Mar 21 17:56:24 CET 2012
[INFO] Final Memory: 9M/152M
[INFO] ------------------------------------------------------------------------
```

Scopes: okruh závislostí

- sú závislosti, ktoré treba pri kompilácii
- sú závislosti, ktoré treba len pri teste
- sú zavislosti, ktoré stačia počas behu
- sú voliteľné závislosti
- …

scopes!

Scopes: okruhy závislosti

```
<project … >
  …
  <dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
  </dependencies>
    …
</project>
```

závislosť je treba ku kompilácii, za behu ju dodá kontajner

Scopes

compile
- potrebné ku skompilovaniu

provided
- compile + závislosť počas behu dodá kontajner

runtime
- len počas behu

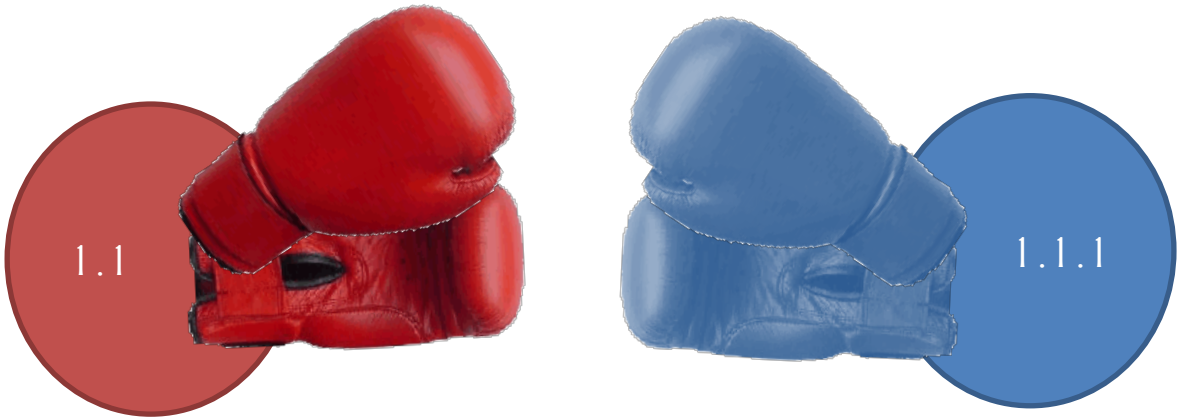
test
- pre kompiláciu a beh unit testov

plus „system" a „import"

Konflikty a.k.a. dependency hell

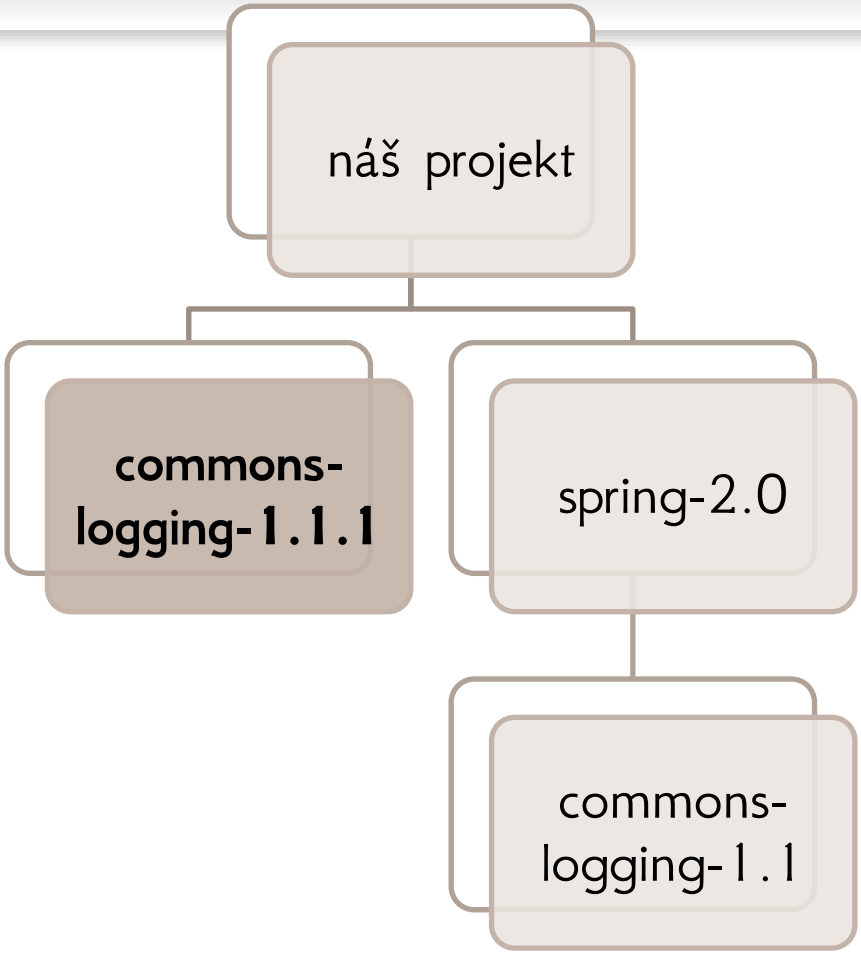
- náš projekt potrebuje commons-logging 1.1.1
- náš projekt závisí na module, ktorý závisí na staršom commons-logging 1.1

konflikt!



Vyhodnotenie konfliktu

- Maven vybuduje strom závislostí
- vezme tú verziu bližšie ku koreňu
- platí pre štandardný prípad



http://guntherpopp.blogspot.com/2011/03/understanding-maven-dependency.html

Konflikty a.k.a. dependency hell

- náš projekt potrebuje Spring
- Spring potrebuje commons-logging
- commons-logging je divný, máme miesto neho slf4j

Vyšmarenie závislosti

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>3.1.1.RELEASE</version>
  <exclusions>
    <exclusion>
      <artifactId>commons-logging</artifactId>
      <groupId>commons-logging</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

odrež vetvu stromu

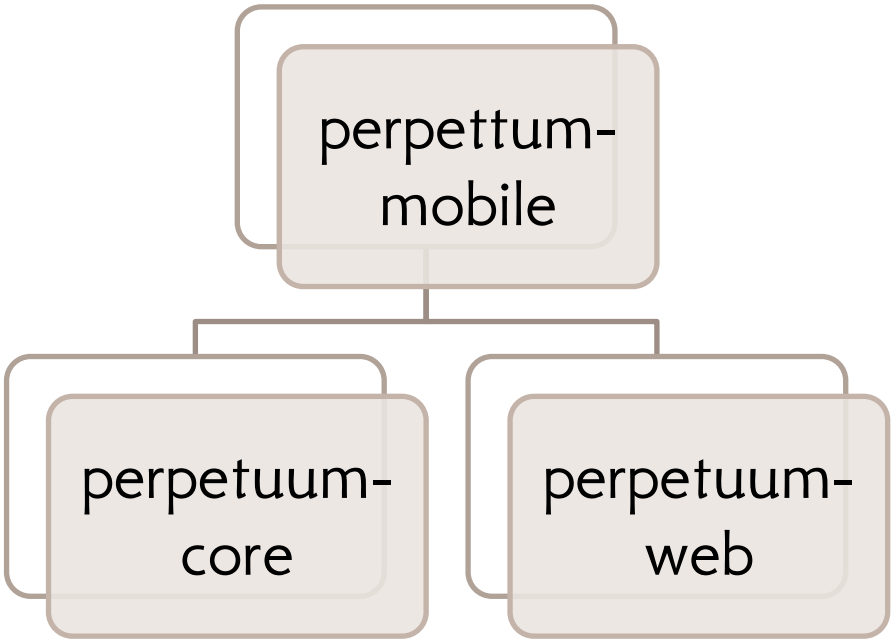
explicitný zákaz tranzitívnej závislosti

multi-module projects

VIACMODULOVÉ PROJEKTY

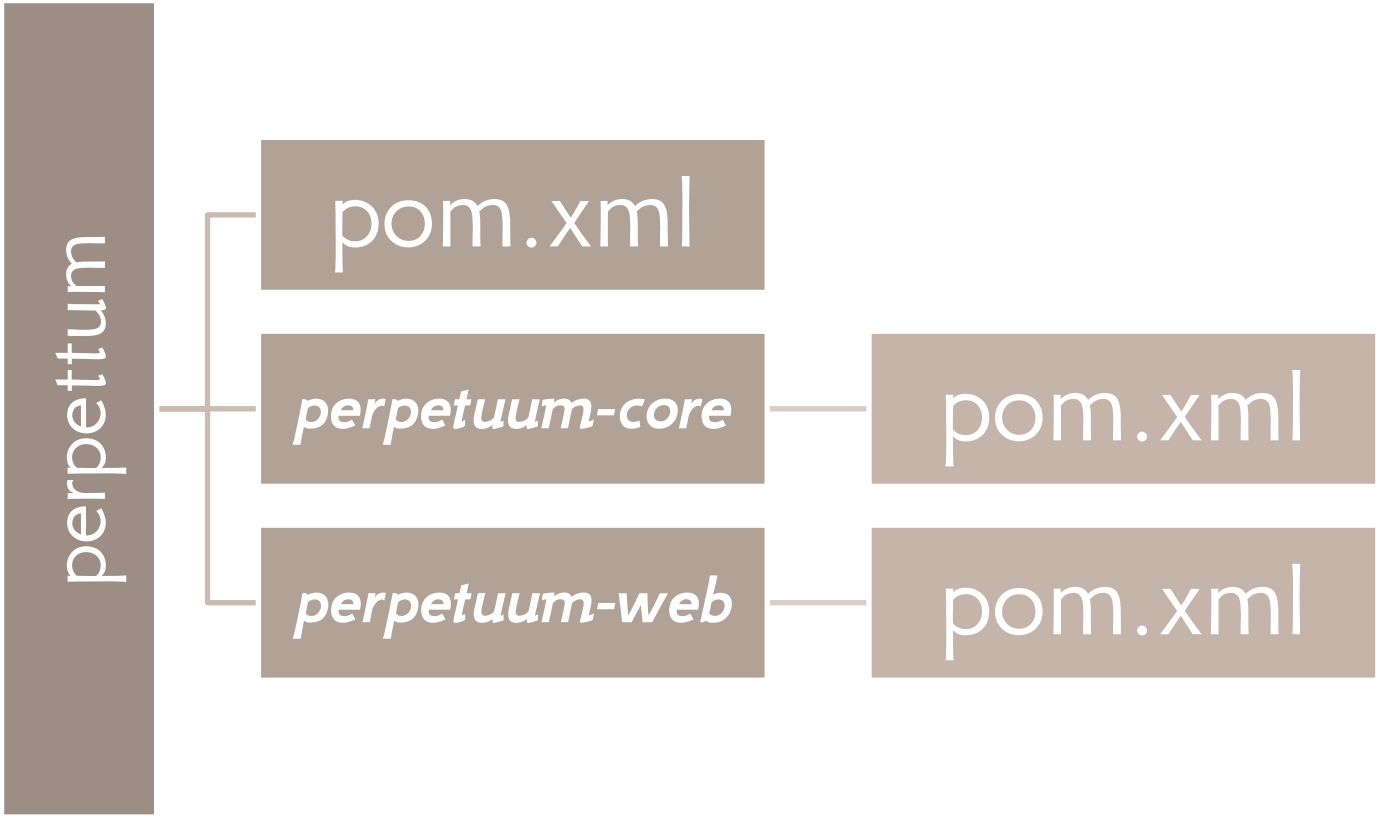
Viac modulov = viac ~~chaosu~~ poriadku

- projekt môže pozostávať z viacerých modulov
- ako ich zostaviť?



Adresárová štruktúra

- každá škatuľa má svoj **pom.xml**



pom pre rodičovský modul

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

  <groupId>sk.upjs.ics</groupId>
  <artifactId>perpetuum-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>perpetuum-core</module>
    <module>perpetuum-web</module>
  </modules>
</project>
```

výsledkom buildovania rodiča je len POM

zoznam modulov

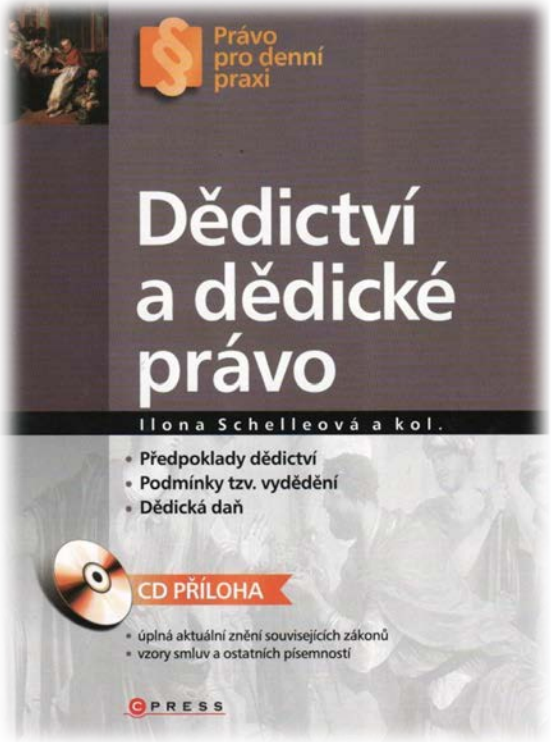
Zostavenie celého projektu

mvn package

- usporiada moduly podľa vzájomných závislostí
  - ak **perpetuum-web** má závislosť na **perpetuum-core**, zoradí ich správne
- na každom vykoná **mvn package**

Dedenie vlastností v moduloch

- v rodičovskom pom.xml možno definovať vlastnosti, ktoré moduly zdedia
- dedia sa
  - číslo verzie
  - závislosti
  - zoznam vývojárov
  - použité plug-iny + konfigurácia



pom pre modul dieťata

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

  <groupId>sk.upjs.ics</groupId>
  <artifactId>perpetuum-core</artifactId>
  <packaging>jar</packaging>

  <parent>
    <groupId>sk.upjs.ics</groupId>
    <artifactId>perpetuum-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  ...
```

dieťa zdedí verziu z rodiča

dedičnosť z rodičovského POMu

Chceme slf4j do všetkých potomkov?

```
<project ...>
  <groupId>sk.upjs.ics</groupId>
  <artifactId>perpetuum-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <dependencies>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>1.6.4</version>
    </dependency>
  </dependencies>
```

Rodičovský POM

Všetky podmoduly získajú závislosť na slf4j-api

Dedičnosť-nededičnosť

- čo ak nechceme zaviesť **tvrdé závislosti** do všetkých rodičov?
- chceme však držať **konzistentnosť**
- v rodičovi definujeme typické závislosti aj s verziami
- podmodul si môže **vybrať**, ktoré použije
  - nemusí však uviesť verziu; tá sa zdedí

Konzistentnosť verzií medzi potomkami

```
<project ...>
  <groupId>sk.upjs.ics</groupId>
  <artifactId>perpetuum-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.6.4</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
...
```

Rodičovský POM

Modul môže využiť závislosť na slf4j-api

pom pre modul dieťata

```
<project>
  ...
  <parent>
    <groupId>sk.upjs.ics</groupId>
    <artifactId>perpetuum-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <dependencies>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
    </dependency>
  </dependencies>
```

POM podmodulu

modul si vyberie závislosť od rodiča

verziu zdedí od rodiča

Sumarizácia modulov

Dedičnosť

- rodič musí mať packaging **pom**
- dieťa dedí cez odkaz v <parent>
- dedí závislosti, verziu…

Kompozícia

- rodič uvedie zoznam modulov v <modules>
- spustenie goalu / fázy sa propaguje do podmodulov

Radostne možno kombinovať!

Rodičovský POM má zoznam modulov, moduly od neho dedia

eclipse / netbeans integration

INTEGRÁCIA S IDE [ECLIPSE]

Príklad: integrácia s Eclipse

mvn eclipse:eclipse

- vytvorí konfiguráky pre eclipsácky projekt
  - .project, .classpath
- závislosti špecifikované premennou **M2_REPO**
  - nutné nastaviť v Eclipse
- pri aktualizácii POM treba spustiť goal nanovo

m2eclipse: plug-in

- alternatívne: plug-in pre Eclipse

http://www.sonatype.org/m2eclipse

- GUI pre závislosti, editáciu POMov, spúšťanie goalov…

m2eclipse: plug-in

MessageAnalyzer.java | HandshakeType.java | InputStreamUnwrappin | camel-xmpp-example/p

Overview

Artifact

Group Id: sk.upjs.ics.novotnyr

Artifact Id: * camel-xmpp-example

Version: 1.0-SNAPSHOT

Packaging: jar

Parent

Properties

project.build.sourceEncoding : UTF-8

Create...

Remove

Modules

New module element

Project

Name: camel-xmpp-example

URL: http://maven.apache.org

Description:

Inception:

Organization

SCM

Issue Management

Continuous Integration

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Ďalšie možnosti: projektový web

- generovanie projektovej stránky

```
mvn site
```

- JavaDoc
- report code coverage,
- výsledky testov

Ďalšie možnosti: AntRun

http://ant.apache.org

- alternatívny nástroj pre zostavovanie
  - bližší **make**
- tiež konfigurovaný v XML
- procedurálne orientovaný
  - Maven: viac deklaratívny

Maven vie volať Ant!

Ďalšie možnosti: AntRun

- stačí zadeklarovať plug-in AntRun

```
<plugin>
  <artifactId>maven-antrun-plugin</artifactId>
  <version>1.5</version>
  <configuration>
    <target>
      <echo message="Ahoj svet!"/>
    </target>
  </configuration>
</plugin>
```

Klasické vnútro antovského targetu

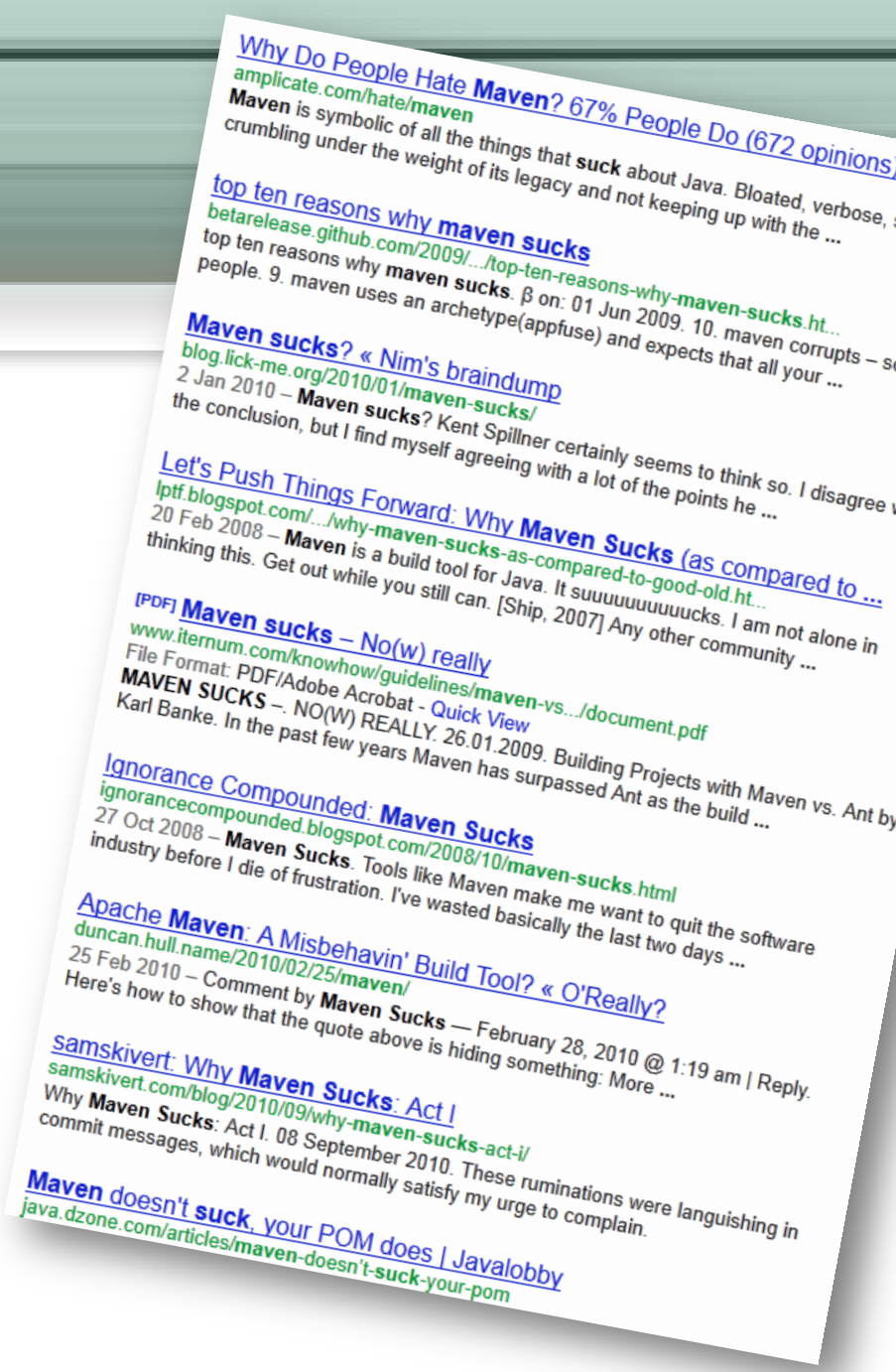
Ďalšie možnosti: AntRun

mvn antrun:run

- spustí antovský target definovaný v konfiguráku
- plug-in možno viazať s konkrétnymi fázami
- vhodné pre úlohy, ktoré sú v Ante, ale Maven nemá pre ne plug-iny

Dosť chválenia!

- **ukecané XML:**
  - 200 riadkové pomy
  - plug-in naviazaný na fázu s konkrétnym goalom a konkrétnou konfiguráciou: 20 riadkov v piatich vnoreniach
- **niekedy WTF správanie**
  - google „Maven Sucks"
  - vyplýva z ukecanosti



Dosť chválenia!

- **nemožnosť skriptovania**
  - ak nestačia plug-iny, nedá sa skriptovať
  - jedine cez AntRun
  - alebo cez vývoj vlastných pluginov
- **1 modul = 1 artefakt**
- **zoskupovanie závislostí**
  - jedine cez vytvorenie separátneho POMu

„Zostavovania v Mavene sú opakovaným zúfalstvom, ktoré vás tiahne do najhlbších priepastí pekelných (v ktorých bol sám Maven stvorený)"

http://kent.spillner.org/blog/work/2009/11/14/java-build-tools.html

Kam inam?

Ant [build] + Ivy [dependencies]

divná dokumentácia

Gradle: deklaratívny, využíva Groovy namiesto XML

chýbajú plug-iny

Buildr: Ruby namiesto XML

vraj pomalší vývoj + moduly z Ruby

Otázky?