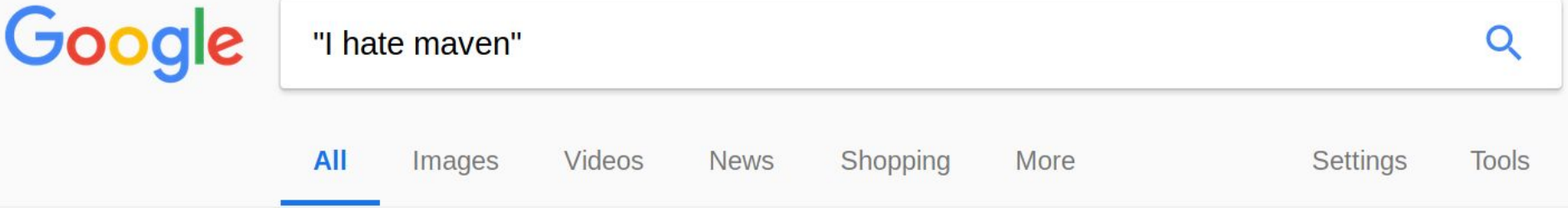
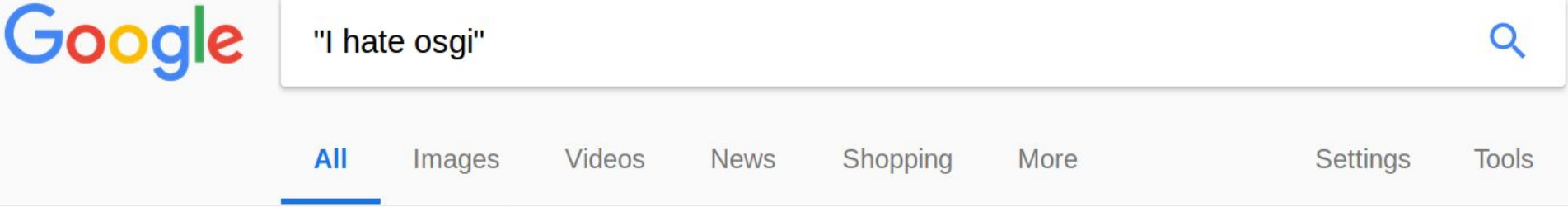
OSGi with Maven

Improved developer productivity thanks to standard tools

Thank you!

To correct me:
luke | code-house.org
twitter: @ldywicki
code: https://github.com/Code-House/eclipsecon (under ASL 2.0)


Google
"I hate maven"
All
Images
Videos
News
Shopping
More
Settings
Tools
About 12,400 results (0.44 seconds)
Honestly… I hate Maven | Virgo's Naive Stories
https://virgo47.wordpress.com/2011/03/07/honestly-i-hate-maven/
Honestly… I hate Maven. March 7, 2011 17 Comments. And I don't give a damn that I don't know it good enough. Why “good enough” in Maven is so difficult ...


Google
"I hate osgi"
All
Images
Videos
News
Shopping
More
Settings
Tools
8 results (0.18 seconds)
Ricardo Argüello on Twitter: "I hate OSGi."
https://twitter.com/ricardoarguello/status/608692105653940224
Jun 10, 2015 - Ricardo Argüello · @ricardoarguello. Un pesimista feliz. Quito, Ecuador. Joined November 2008. Tweets. © 2018 Twitter; About · Help Center ...

L

W OSGi n be not so great..

Increas any 3rd party library integration (like a tax).

Local development turnaround productivity is relatively poor (MVN).

Ask anyone who eve SingleFeatureTest (SFT) failure? ;)

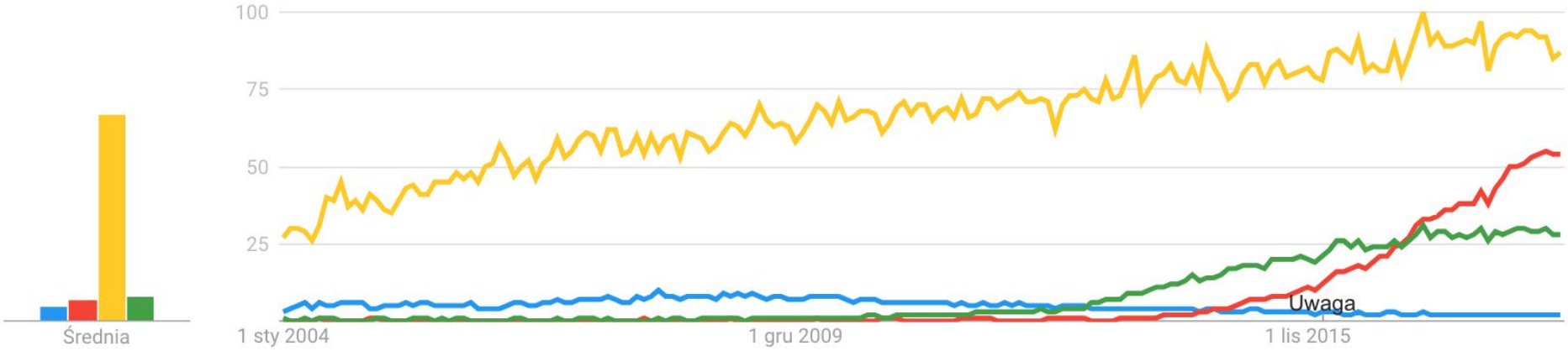
It's basically just re "niche" the overall Java ecosystem. Check SO.

This costs us real pro e and money overhead, IMHO.

OPENDAYLIGHT

Source: https://twitter.com/vorburger/status/1044203455055892481

Google trends



About me

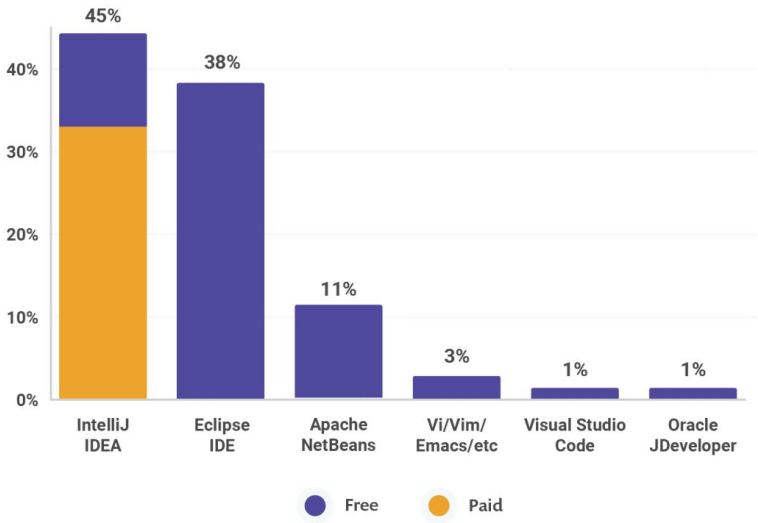
Self employed consultant since 2008, a middleware specialist.

- Open Source enthusiast since forever
- Apache Karaf
  - committer since 2010
  - project management committee since 2012
- I worked for
  Red Hat, FUSE Source, been involved in projects for Alcatel-Lucent, Nokia Siemens Networks, Polish Post Bank, Royal Bank of Scotland etc.
- Eclipse user since 2003
- Maven user since 2006

Few words about world

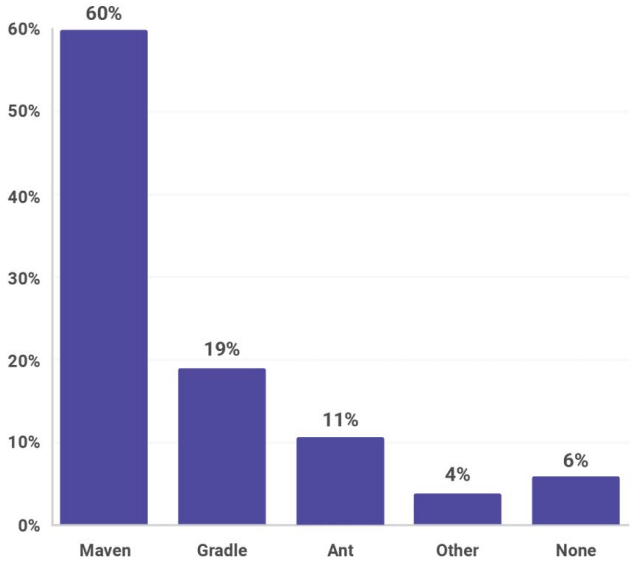
How does it look a like outhere

Tool usage



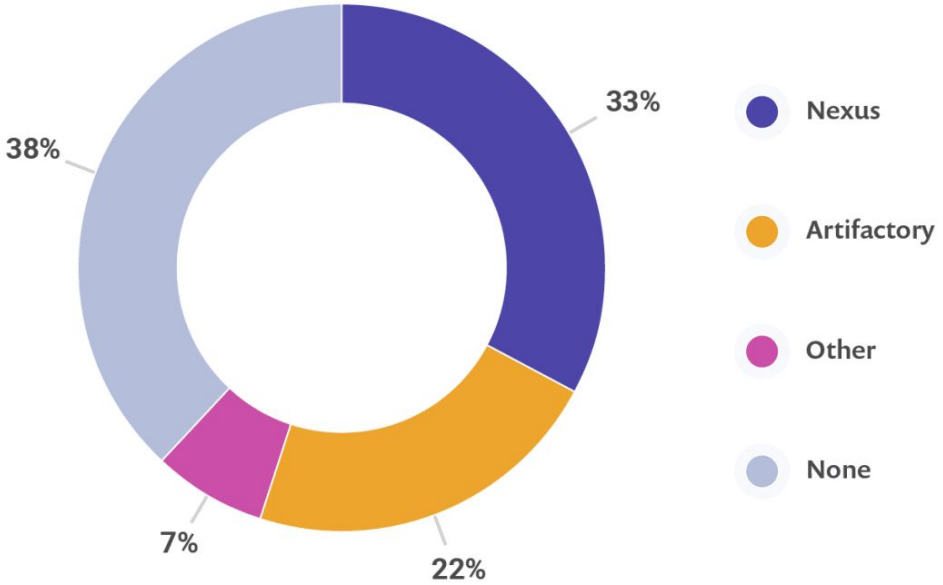
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Build tools



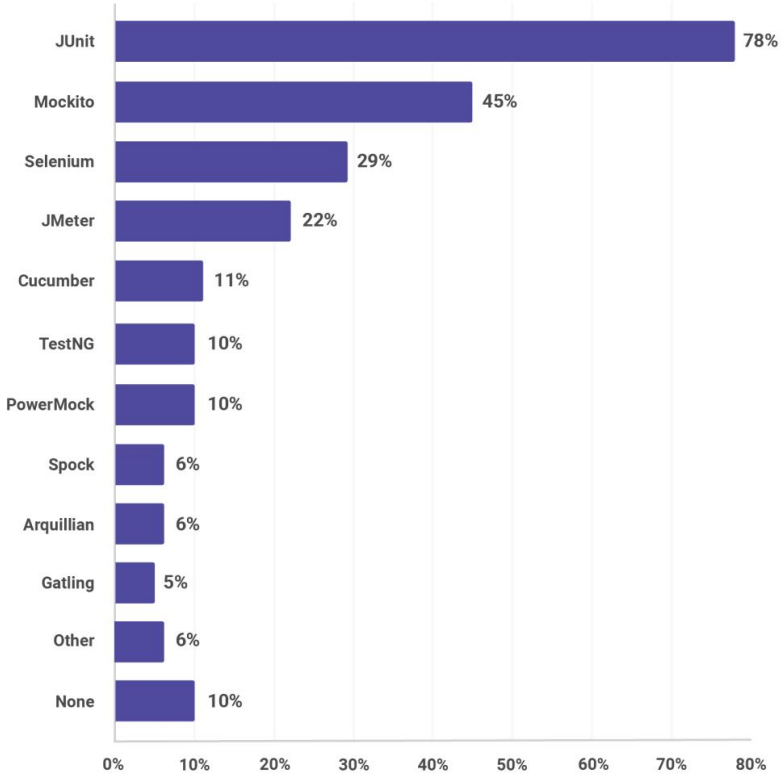
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Artifact repositories



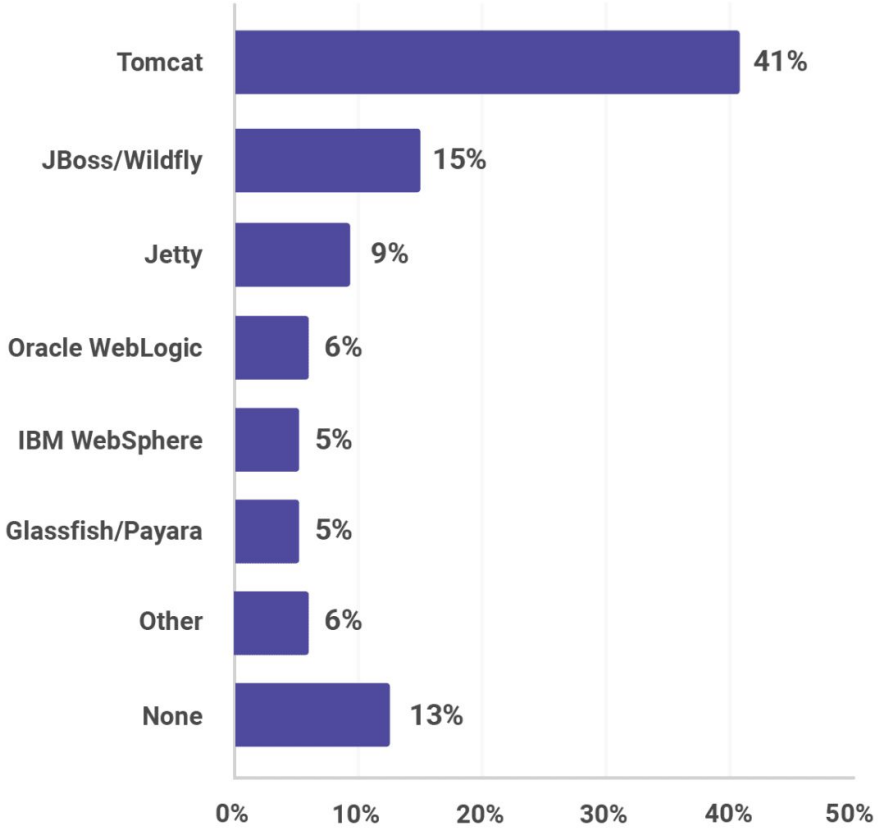
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Testing tools



Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Runtimes



Source: https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application

OSGi is not classified in runtimes

- Some of server runtimes can run OSGi
- OSGi can be run inside a WAR
- There are a lot of people who knows Maven, Tomcat, IntellIJ and Eclipse and development model promoted by these tools

Communication troubles

Pay attention to what you say

There is a problem finding people knowing OSGi.

Closer look on Maven

Maven is ...

- build tool
  - compiler
  - test runner
  - packager
  - site generator

But it is also

- Tool promoting convention over configuration
- Repository client
- A dependency manager
- A pluggable machina

a word about configuration

- Global
  $M2_HOME/conf/settings.xml
- Local, user settings
  ~/.m2/settings.xml
- Project
  pom.xml

a bit about inheritance

- Projects can be organised into a tree:
  - super pom (implicit)
    - organisation POM
      - project A
        - child project 1
        - child project 2
      - project B
        - first child
        - second child

what is inherited

- dependency management
- dependencies
- plugins
  - executions
  - configurations
- resources configuration

Convention over configuration

- You palce sources in src/
  - code in main/java, resources in main/resources
  - tests in test/java
- Results end up in target/
- There are standard build phases (pre-test, test, package etc.)
- Plugins are hooked into phases
- Depending on plugin developer, there are various defaults

Repository client

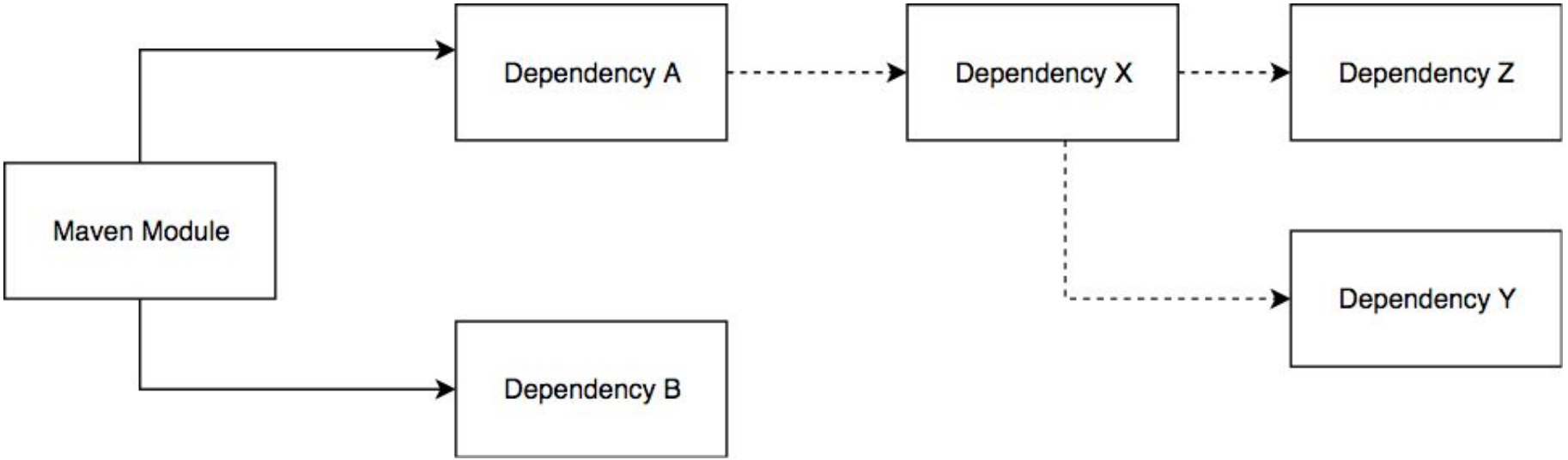
- Maven brought remote repository idea to mainstream
- Elements are identified by:
  - group id
  - artifact id
  - version
  - classifier
  - type (packaging)

  Location: ${group id}/${artifact id}/${version}/${artifact id}-${version}[-${classifier}].${type}
- One project may have multiple repositories
- The central.maven.org is enabled by default
- You are free to define own repositories
  - in project descriptor
  - local or global maven settings

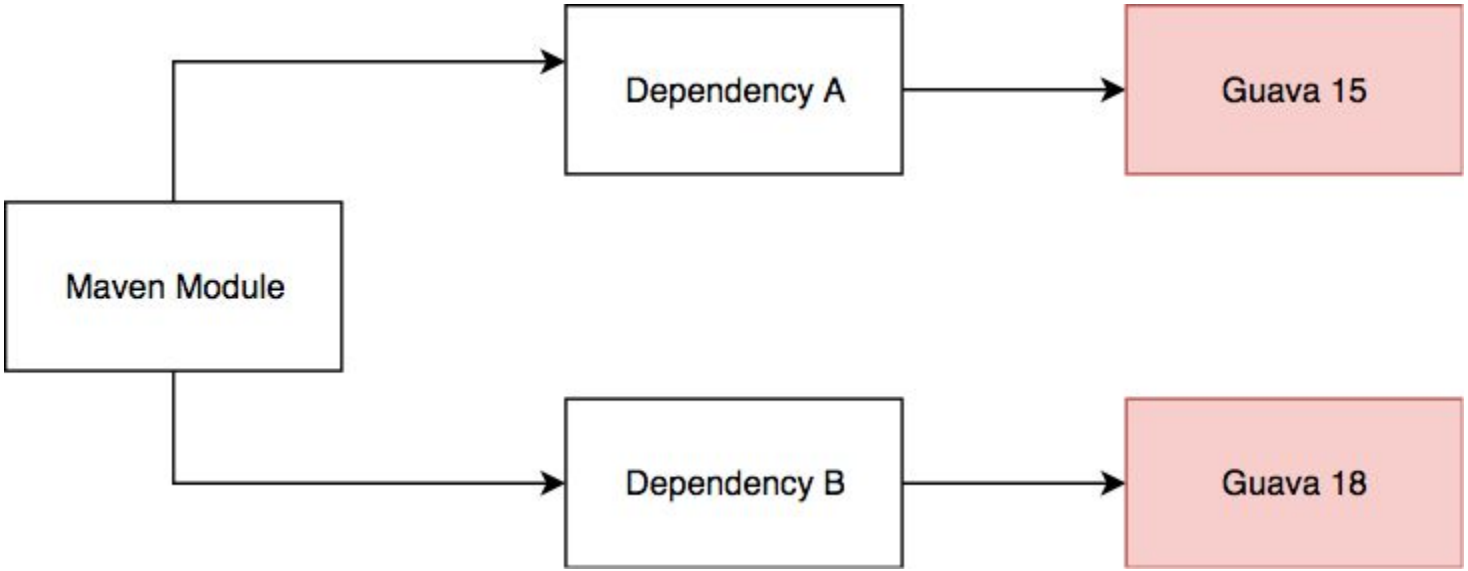
Dependency manager

- Maven recognizes basic dependency scopes
  - compile
  - runtime
  - test (not transitive)
  - provided (not transitive)
- system / import scopes are special kind
- Dependencies can be managed / forced by parent and also overridden

Transient dependencies



What dependency management do



#include <dependency.h>

```
<dependencyManagement>
 <dependency>
  <groupId>org.eclipse.smarthome</groupId>
  <artifactId>smarthome</artifactId>
  <version>0.10.0-SNAPSHOT</version>
  <scope>import</scope>
  <type>pom</type>
 </dependency>
</dependencyManagement>
```

Pluggable machina

- Maven internally uses simple IoC based on classpath scanning
- Classpath can be extended via plugin dependencies & extensions
- There is whole new extension mechanism in 3.x (enabling polyglot maven)
- Almost all, if not all, components are injected via IoC
- … did I mention that maven supports version ranges?

Versions and ranges under Maven

- 2 becomes 2.0.0 when comparing
- Special values in qualifier:
  - "alpha" < "beta" < "milestone" < "cr" / "rc" < "snapshot" < "final" / "ga" < "sp"
- 2.0-Snapshot matches [1,2.0) boundary (excludes 2.0 release)
- Customisable via SPI:
  - Version Scheme
  - Version Range Resolver
  - OSGi compatible version: https://github.com/Code-House/maven-osgi-resolver

Commonalities

Comparison

Element	OSGi	Maven
Build unit	Bundle	Project / Module
Dependency unit	Package, req /cap	Dependency
Platform definition	Target Platform (PDE)	Dependency management
Version ranges	Yes	Yes
Repositories	OBR, P2	Maven repos

Project organisation

Systems are like onions

Source: http://www.downvids.net/shrek-moments-quot-ogres-are-like-onions-quot--317773.html, Shrek movie was made by DreamWorks

Each bigger system have

- Domain model
- Persistence
- Communication interface
- Many provider/consumer relationship between these parts

Properly structured project is easier to run and maintain.

Most of OSGi projects is properly structured.

Some Maven based projects have good structure.

Basic project structure

- root
  - model
  - api
  - core
  - dao
    - api
    - core
  - web
    - api
    - core
  - features

Manifest is just another resource

How to generate manifest

```
<packaging>bundle</packaging>
<!-- ... -->
<plugin>
 <groupId>org.apache.felix</groupId>
 <artifactId>maven-bundle-plugin</artifactId>
 <configuration>
  <instructions>
   <Export-Package>org.code_house.eclipsecon.mvnosgi.api</Export-Package>
   <Private-Package>org.code_house.eclipsecon.mvnosgi.core</Private-Package>
   <Bundle-Activator>org.code_house.eclipsecon.mvnosgi.core.Activator</Bundle-Activator>
  </instructions>
 </configuration>
</plugin>
```

bndlib ✔️

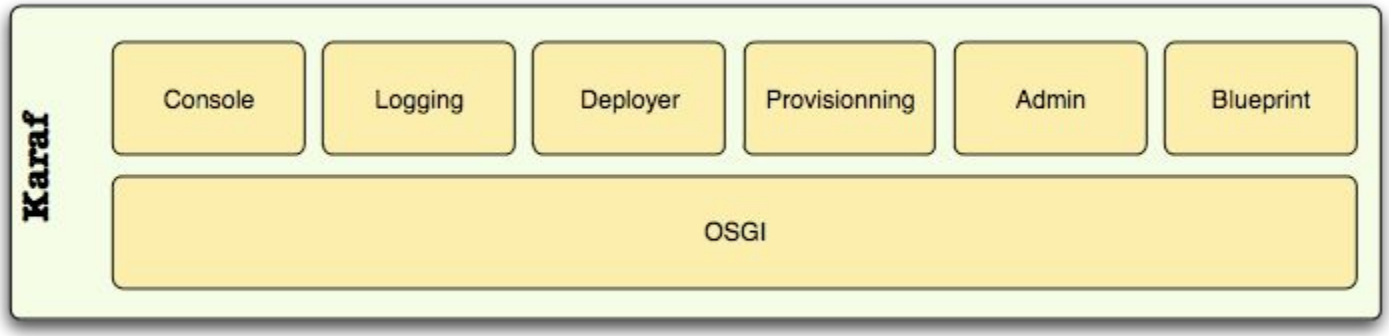
Manifest defaults

Bundle-SymbolicName: ${project.groupId}.${project.artifactId}
Bundle-Version: ${project.version}
Bundle-Name: ${project.name}
Bundle-Description: ${project.description}
Bundle-Vendor: ${project.organization.name}
Bundle-License: ${project.license*.url}
Import-Package: *
Export-Package: *
Private-Package: *.impl.*, *.internal.*

Deployment

How to bring bundles to runtime

Quick look on Karaf



Pax URL

- By default Java runtime supports following URI schemes:
  - http[s]:
  - file:
- Java / OSGi framework allows to provide new schemes (ie. bundle:)
- Pax URL provides:
  - wrap:
  - classpath:
  - mvn:
  - and few more (exploded dir and similar)

Deploying bundles

- install mvn:org.code-house.eclipsecon.mvnosgi/api/RELEASE
- install mvn:org.code-house.eclipsecon.mvnosgi/core/RELEASE
- install mvn:org.code-house.eclipsecon.mvnosgi.dao/api/RELEASE
- install mvn:org.code-house.eclipsecon.mvnosgi.dao/core/RELEASE/jar
- install mvn:org.code-house.eclipsecon.mvnosgi.web/api/LATEST/x86_64/jar

Karaf feature

```
<feature name="book-service" version="${project.version}">
        <feature>scr</feature>
        <feature>cxf-jaxrs</feature>
        <feature>aries-blueprint</feature>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi/core/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi/api/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi/model/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi.dao/api/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi.dao/core/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi.web/core/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi.web/api/1.0.0-SNAPSHOT</bundle>
        <bundle>mvn:org.code-house.eclipsecon.mvnosgi/dto/1.0.0-SNAPSHOT</bundle>
        <!-- ... rest of stuff ... -->
```

Testing

How to bring bundles to runtime

OSGi -> junit -> test

junit -> OSGi -> test

Demo time

Arquillian

Pax Exam

Pax Tinybundles

```
InputStream inp = bundle()
  .activator(TestActivator.class)
  .add(BookService.class)
  .add(TestBookService.class)
  .symbolicName("test-bundle")
  .set(Constants.EXPORT_PACKAGE, "demo")
  .set(Constants.IMPORT_PACKAGE, "demo")
  .build();
```

Exam containers

- OSGi
- Karaf
- Tomcat
- Weld
- JBoss / Wildfly 8 / Wildfly 9

Example test

```
@RunWith(PaxExam.class)
@ExamReactorStrategy(PerMethod.class)
public class ExampleIntegrationTest {
 @Configuration public Option[] config() {
  return options(junitBundles());
 }
 @Test …
}
```

Unit testing - Felix Connect

- Emulates OSGi framework
  - Activators
  - Service lookups
  - Bundles and classloaders
- Perfect candidate for unit testing based on plain maven classpath

Felix Connect

```
String bundleFilter = "(&(Bundle-SymbolicName=*)(!(Bundle-SymbolicName=org.osgi.*)))";
List<BundleDescriptor> descriptors = new ClasspathScanner().scanForBundles(bundleFilter, loader);

// setup felix-connect to use our bundles
Map<String, Object> config = new HashMap<>();
config.put(PojoServiceRegistryFactory.BUNDLE_DESCRIPTORS, bundles);
PojoServiceRegistry reg = new PojoServiceRegistryFactoryImpl().newPojoServiceRegistry(config);
BundleContext bundleContext = reg.getBundleContext();
```

JUnit 5

AssertJ

```
// entry point for all assertThat methods and utility methods (e.g. entry)
import static org.assertj.core.api.Assertions.*;

// basic assertions
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);
```

Questions

Thank you!

To correct me:
luke | code-house.org
twitter: @ldywicki
code: https://github.com/Code-House/eclipsecon (under ASL 2.0)