OSGi + Plain Old Maven

"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 ...

"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

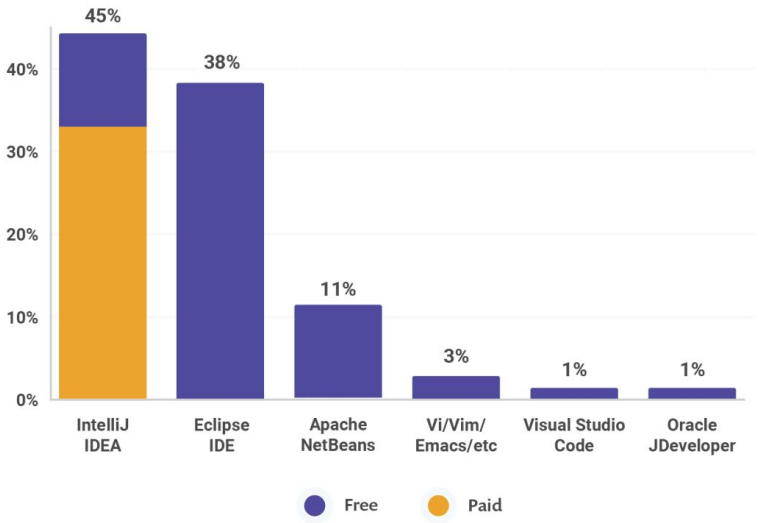
About me

- OpenHab community member
  - BacNet binding
  - Basic karaf support
  - Eclipse SmartHome security
- Java consultant

Few words about world

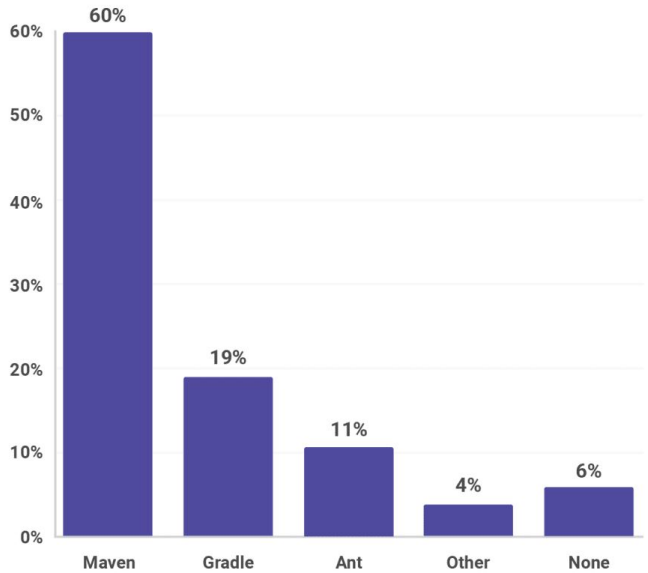
How does it look a like outside

Tool usage



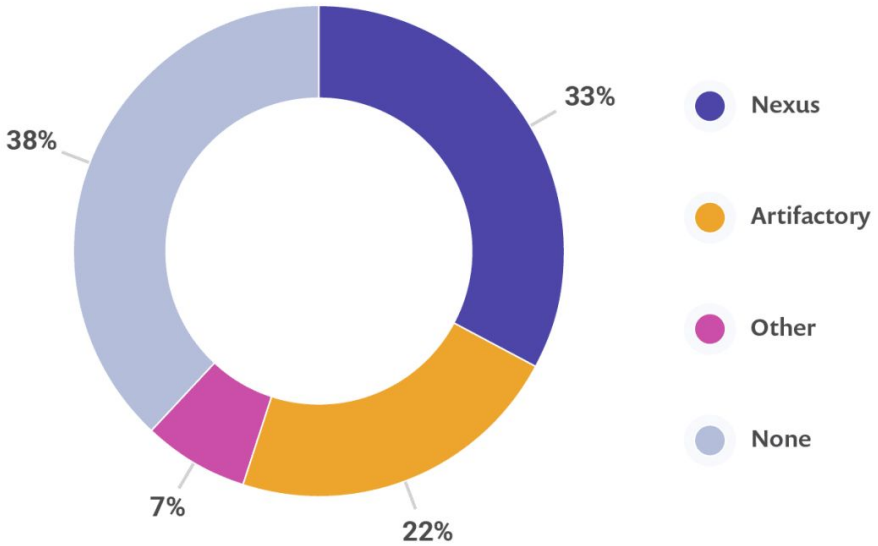
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Build tools



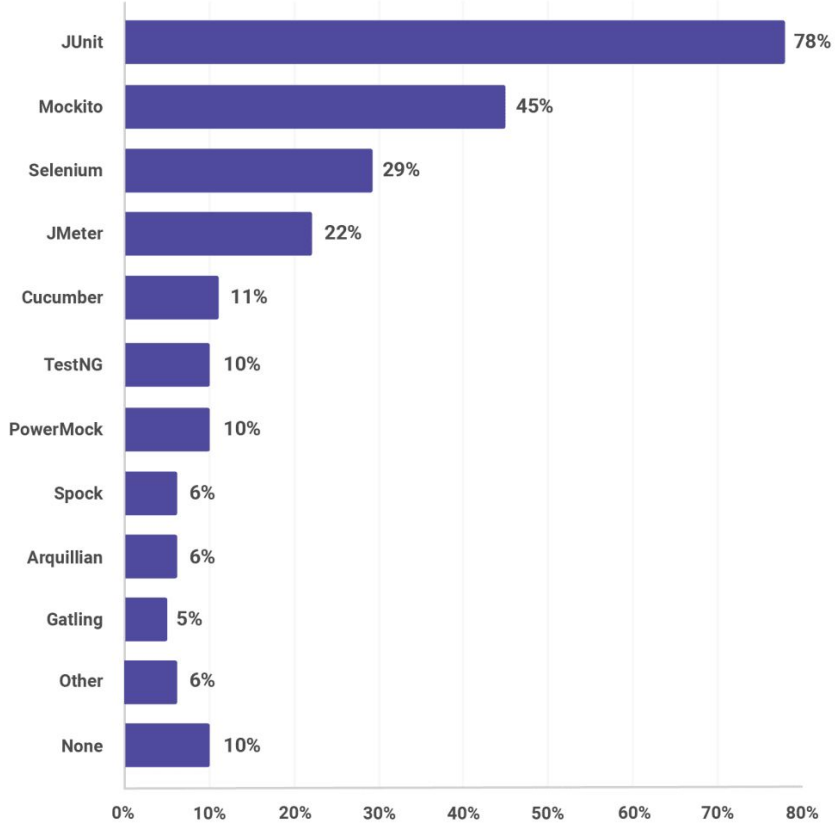
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

Artifact repositories



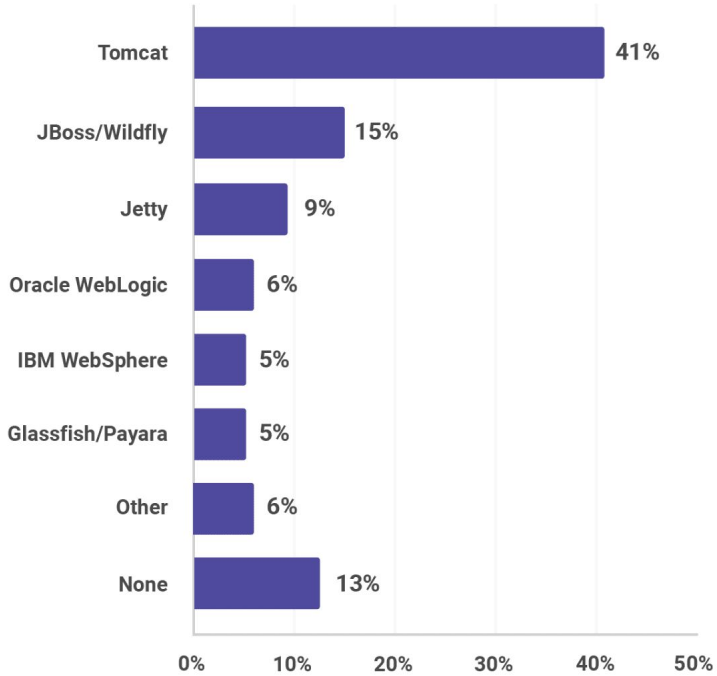
Source: https://snyk.io/blog/jvm-ecosystem-report-2018-tools

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

Ecosystem

- Manifest is useless outside OSGi world
- You need dependency information to compile your project
- Having them in POM allows others to be in sync with them
- BOM - bill of materials pom to import entire dependency management

Issues with Tycho

- There is no offline mode
- Stability of build is questionable
- Slower than normal maven
- "Useless" P2 metadata for primary scenario
- Manually maintained manifest
- Lack of integration with standard repos
- No dependency integration for maven users

#include

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

Any Java developer can be productive with our OSGi stack.

Better support for standard tools, less complains.

Manifest is just another resource

Manifest is just another resource

- Regardless of its importance for OSGi - it is just another descriptor
- Common practice is to generate descriptors from sources/annotations
- Maintaining manifest is waste of time which kills productivity

Manifest is just another resource

- Eclipse PDE
- Maven + Tycho
- any other reason … ?

How to generate SCR

```
<plugin>
 <groupId>org.apache.felix</groupId>
 <artifactId>maven-scr-plugin</artifactId>
 <configuration>
  <specVersion>1.1</specVersion>
 </configuration>
 <executions>
  <execution>
   <id>generate-scr-scrdescriptor</id>
   <goals>
    <goal>scr</goal>
   </goals>
  </execution>
 </executions>
```

How to generate manifest

```
<packaging>bundle</packaging>
<!-- ... -->
<plugin>
 <groupId>org.apache.felix</groupId>
 <artifactId>maven-bundle-plugin</artifactId>
 <configuration>
  <instructions>
   <Export-Package>org.foo.myproject.api</Export-Package>
   <Private-Package>org.foo.myproject.*</Private-Package>
   <Bundle-Activator>org.foo.myproject.impl1.Activator</Bundle-Activator>
  </instructions>
 </configuration>
</plugin>
```

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: *

Enhanced settings

```
<properties>
 <bundle.namespace>${project.groupId}.${project.artifactId}</bundle.namespace>
 <osgi.import>
  !${bundle.namespace},*
 </osgi.import>
 <osgi.export>
  !${bundle.namespace}.internal.*,
  ${bundle.namespace}.*
 </osgi.export>
 <osgi.private>${bundle.namespace}.internal.*</osgi.private>
</properties>
```

Enhanced settings cd.

```
<configuration>
 <instructions>
  <Import-Package>${osgi.import}</Import-Package>
  <Export-Package>${osgi.export}</Export-Package>
  <Private-Package>${osgi.private}</Private-Package>
  <_removeHeaders>
   Import-Service, Export-Service,
   Include-Resource, Private-Package,
   Embed-Dependency
  </_removeHeaders>
 </instructions>
</configuration>
```

Deployment

How to bring bundles to runtime

Pax URL

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

Deploy bundle

- install mvn:org.code-house.osgi/api
- install mvn:org.code-house.osgi/api/LATEST
- install mvn:org.code-house.osgi/api/RELEASE
- install mvn:org.code-house.osgi/api/1.0.0
- install mvn:org.code-house.osgi/api/1.0.0/jar
- install mvn:org.code-house.osgi/api/1.0.0/tests/jar

Testing

OSGi + Tests = mess

OSGi -> junit -> test

junit -> OSGi -> test

Arquillian

Pax Exam

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 …
}
```

Felix Connect

- Emulates OSGi framework
  - Activators
  - Service lookups
  - Bundles and classloaders
- Perfect candidate for unit testing based on 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);
```

Demo time