holisticon
MEMBER OF SIGMA GROUP

# Builddreikampf: Ant, Maven und Gradle

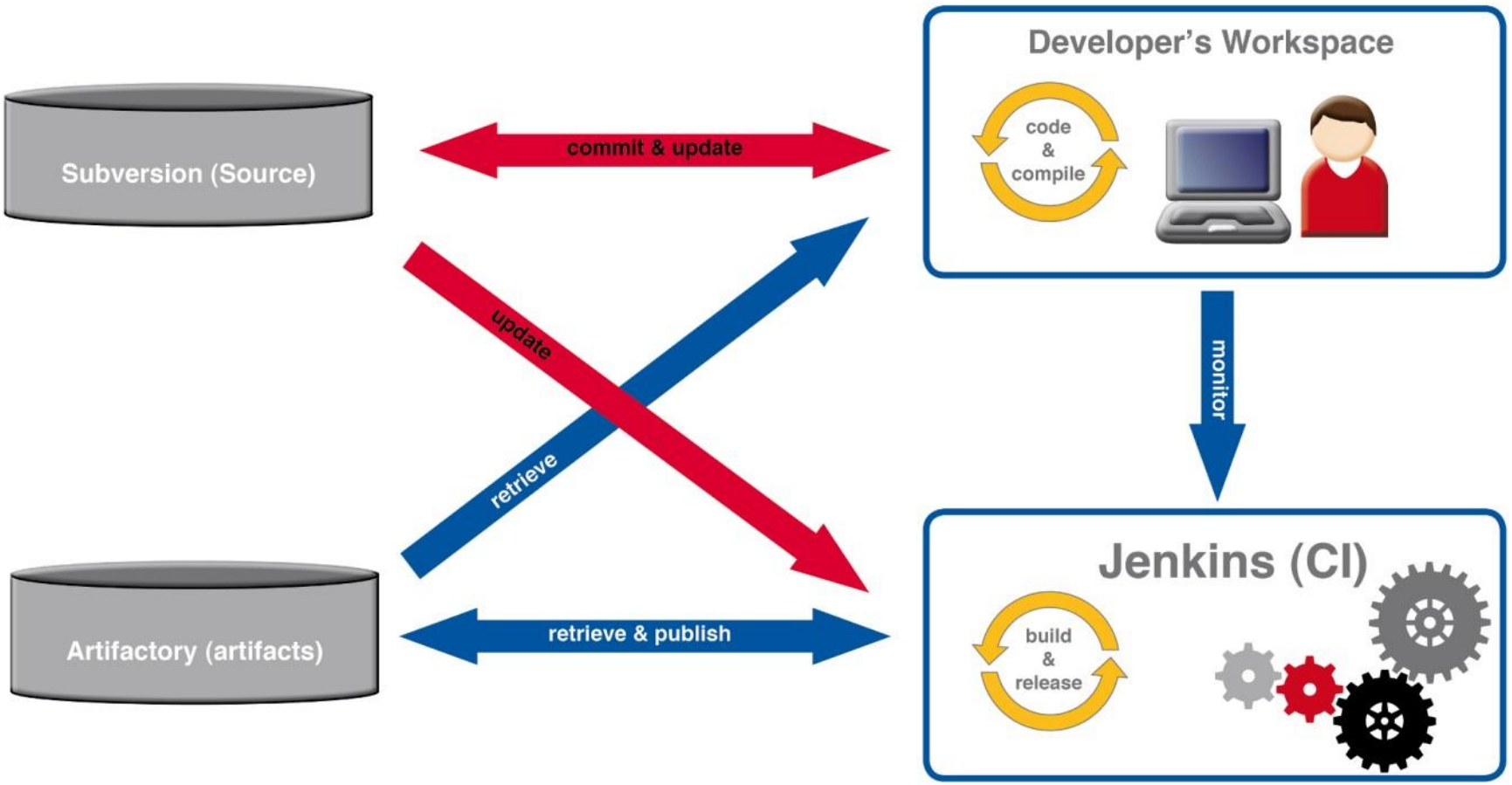
Sven Bunge / Carl Düvel

holisticon AG

# Wettkampfplan

1. Die Regeln
2. Vorstellung der Kandidaten
3. Ring frei – die Disziplinen!
    1. Dependency Management
    2. Multiprojektsupport
    3. IDE-Support
    4. Support
    5. Lizenz
    6. Performance
4. Migrationspfade
5. Hilfestellung zur Toolwahl

# Die Grundregeln



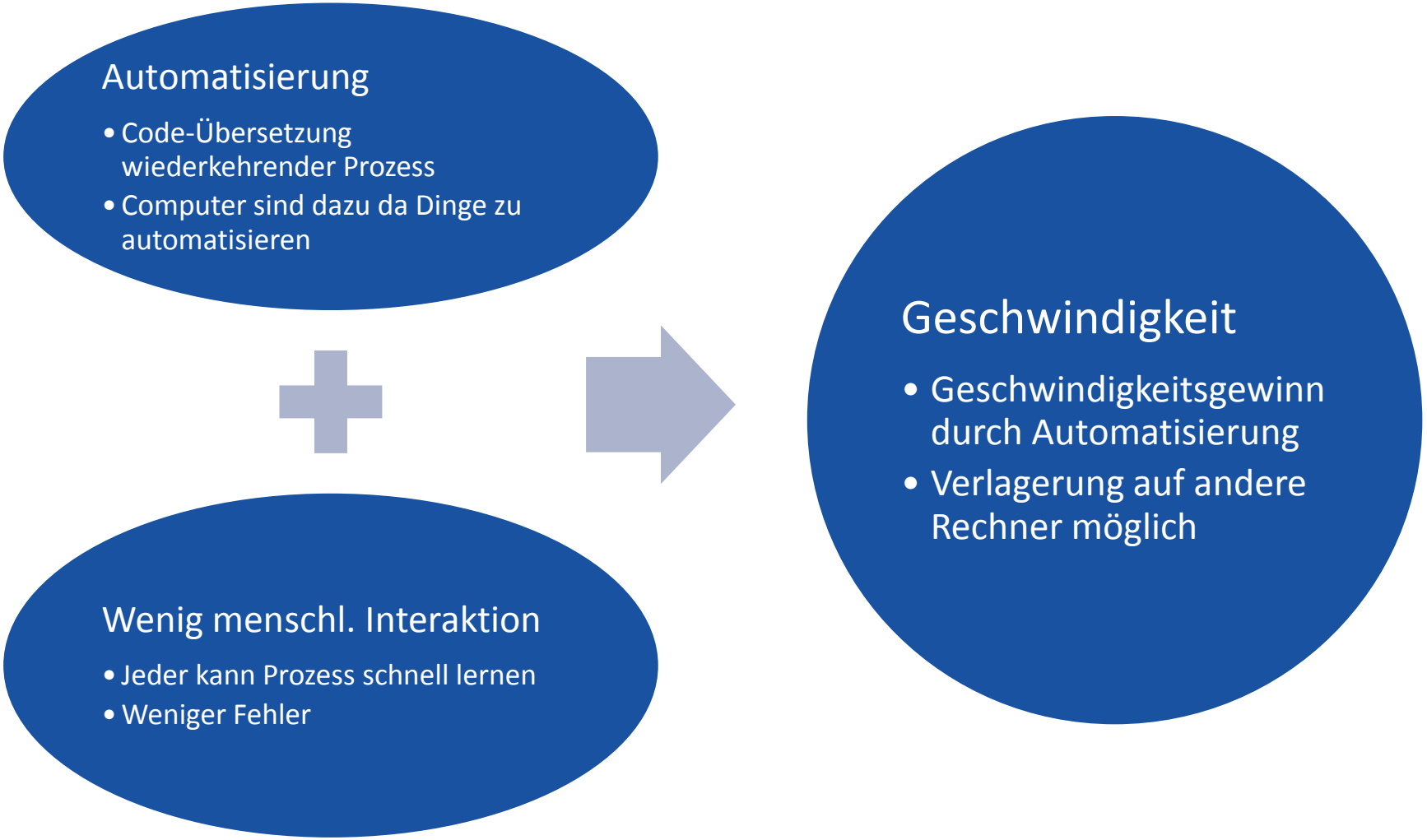
# Anforderung an das Build-System

- Sourcecode bauen und packen
- Testen
  - Automatische Ausführen von Tests
  - Von Unit bis zu Akzeptanz-Tests
  - Codeanalyse
- Dokumentation
  - JavaDoc
  - SourceCode
  - Komplette Projekt-Dokumentation
- Dependency-Management

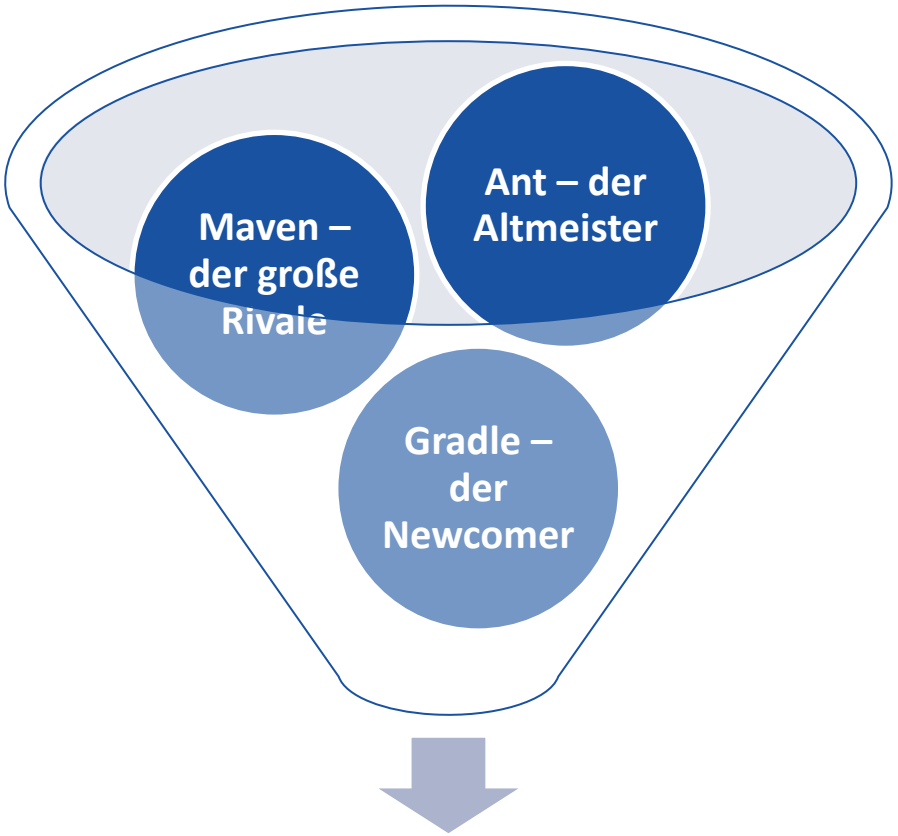
# Weitere Anforderung an das Build-System

- Verteilung der erzeugten Artefakte
- Konfigurations-Management
  - Programme releasen
  - Programm-Versionen sauber verwalten
- Rollout auf Systeme

# Zusammengefasste Anforderungen

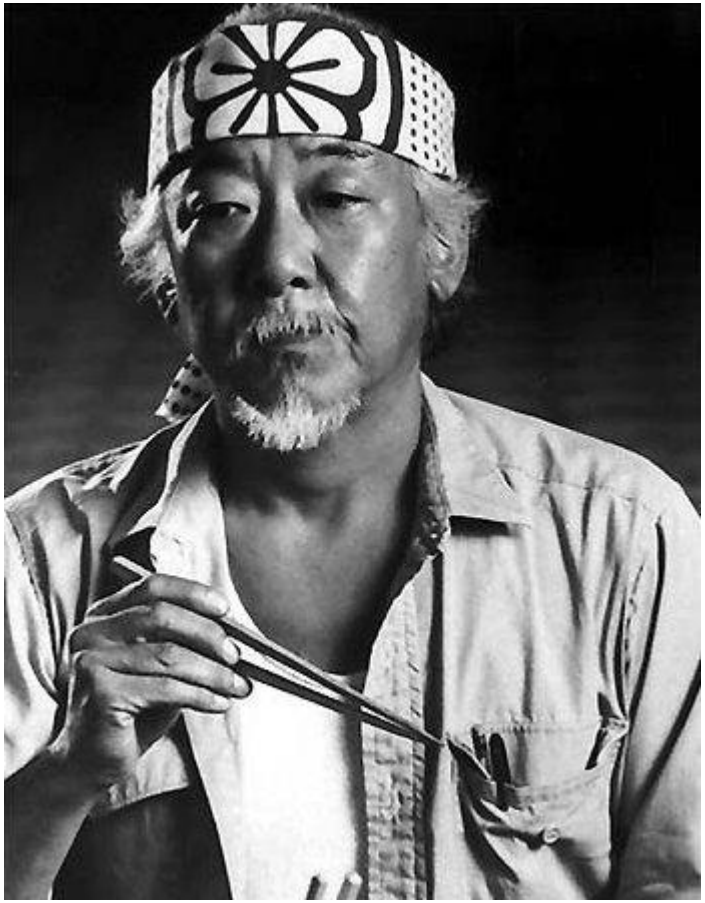


# 2. Die Kandidaten




**Gewinner?**

# Ant – der Altmeister




# Ant – der Altmeister

## Der Urvater: make

```
all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm -rf *o hello
```

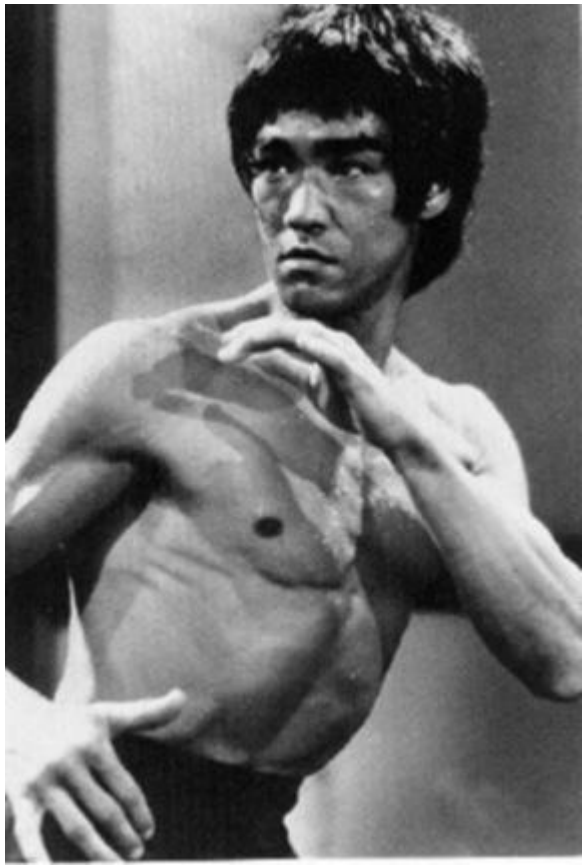
# Ant – der Altmeister

```
<project name="MyProject" default=„compile" basedir=".">
<!-- set global properties for this build -->
 <property name="src" location="src"/>
 <property name="build" location="build"/>

 <target name="init">
  <!-- Create the time stamp -->
  <tstamp/>
  <!-- Create the build directory structure used by compile -->
  <mkdir dir="${build}"/>
 </target>

 <target name="compile" depends="init“ description="compile the source " >
  <!-- Compile the java code from ${src} into ${build} -->
  <javac srcdir="${src}" destdir="${build}"/>
 </target>
</project>
```

# Maven – der große Rivale

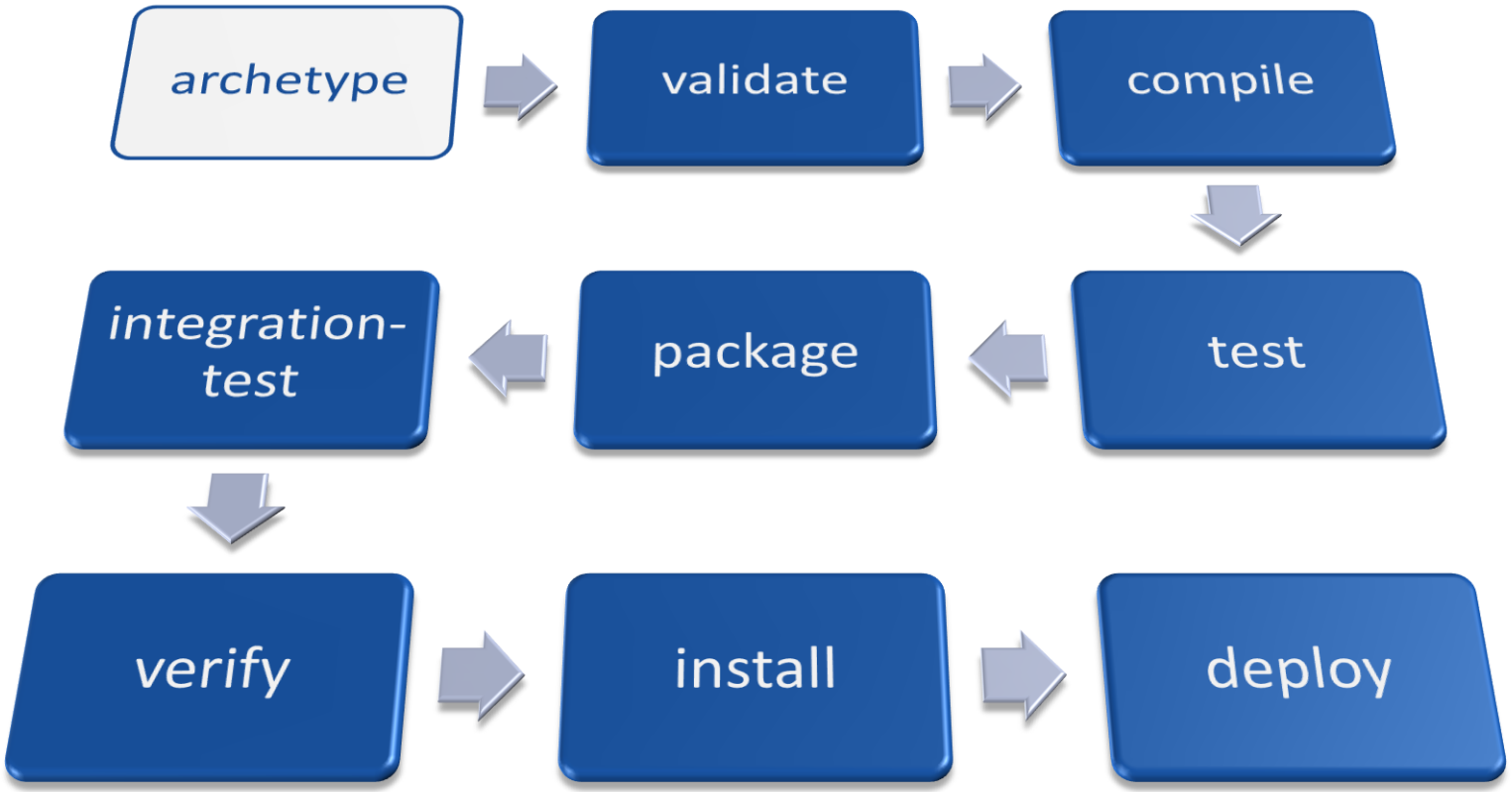


maven

# Maven – Es gibt nur einen Weg!

- Deklarativer Ansatz
- Feste Lifecycle für Aufräumen, Bauen und Dokumentation
- Funktionalität wird durch Plugins bereitgestellt
  - Plugins werden an Phasen im Lifecycle gebunden
  - Plugins auch für Release- und Dokumentations-Prozesse
- Convention over Configuration
  - Vorgaben für alle Details eines Builds: Verzeichnisse, Ablauf, Distribution
  - Nur wer von Vorgaben abweichen möchte, muss die Konfiguration anpassen.
- Dependency-Management
  - Auch Maven-Plugins sind Abhängigkeiten

# Maven - Default-Lifecycle



# Maven-Konfiguration

- Nur eine XML-Datei pro Projekt: *pom.xml*

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.holisticon.builddreikampf</groupId>
  <artifactId>MavenTest</artifactId>
  <version>1.0</version>
</project>
```

- Keine Möglichkeit von XML-Importen
- Einstellungen werden vom Eltern-Projekt übernommen

# Gradle – der Newcomer

# Gradle – Das beste aus beiden Welten?

- Imperativer Ansatz
  - Ant-Targets == Gradle-Tasks
- Groovy-DSL statt XML
- Standard-Plugins für den deklarativen Ansatz
  - Definiert feste Tasks im Build-Lifecycle
- Tasks über Closures erweiterbar
- Ant & Ivy unter der Haube

# Gradle – Simple an simple Ziele

- Einfachste *build.gradle*:

```
apply plugin: 'java'
```

- Zusätzliche Tasks („gradle tasks“)

```
assemble - Assembles all Jar, War, Zip, and Tar archives.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles the main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles the test classes.
```

- Aufruf mit „gradle <task>“

# Gradle – Fremde Tasks gefügig machen

- Kleine Erweiterungen von Plugins

```
apply plugin: 'java'

build.doFirst {
	println "Ich werde nun bauen!!!"
}

build.doLast {
	println "Bob hat fertig gebaut!"
}
```

# Gradles Verbündeter: Ant

- Ant-Build: build.xml

```
<project>
	<target name="AntHello">
		<echo message="I'm ant" />
	</target>
</project>
```

- Gradle: build.gradle

```
ant.importBuild 'build.xml'

task sayHello(dependsOn: AntHello) << {
	println "Ich bin Gradle!"
	ant.echo "Aber ich hab den Ant in mir!"
}
```

# Gradles Verbündeter: Ant – Teil 2

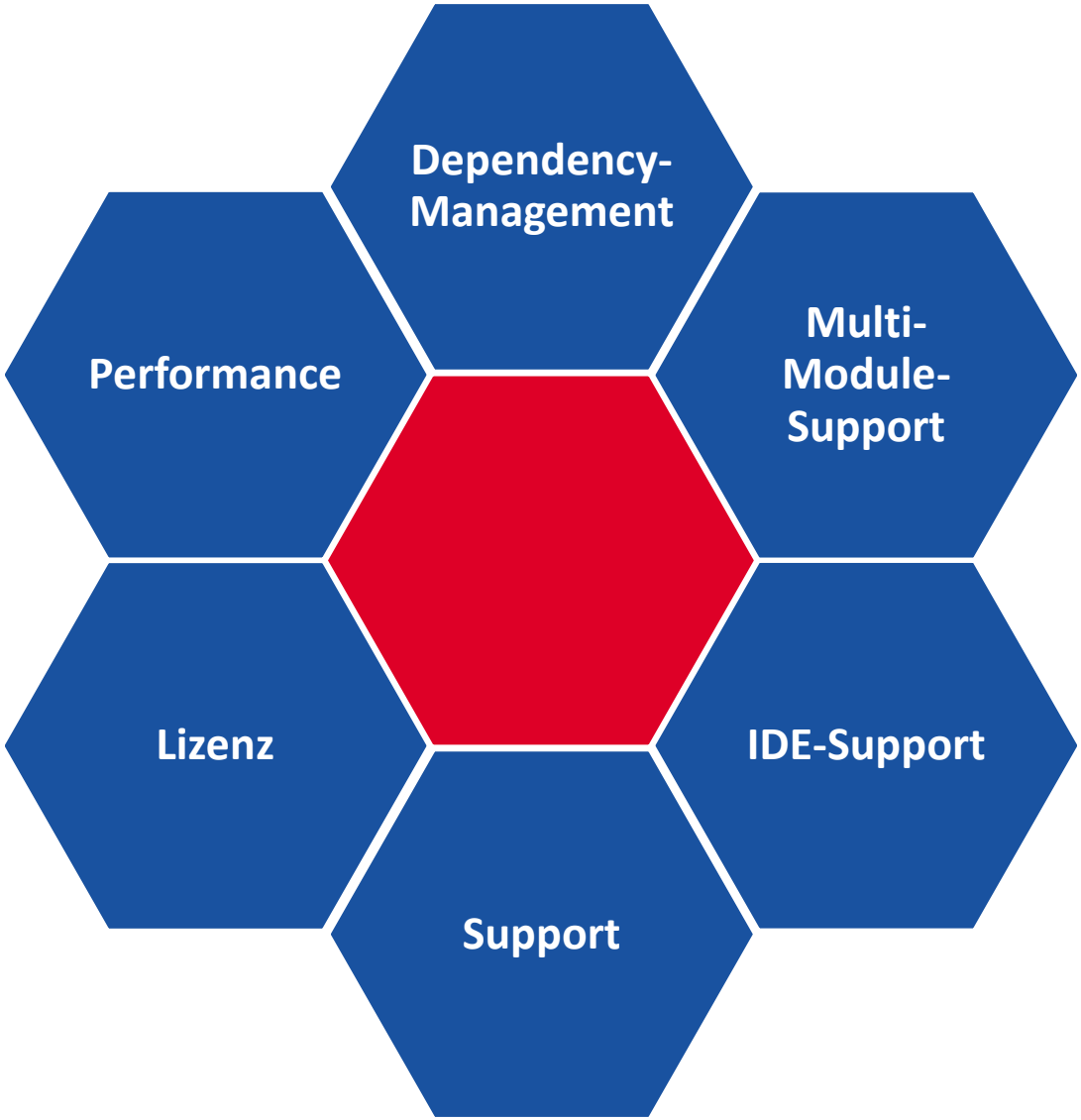
- Aufruf „gradle tasks –all“

```
Other tasks
-----------
sayHello
    AntHello
```

- Aufruf „gradle sayHello“

```
:AntHello
[ant:echo] I'm ant
:sayHello
Ich bin Gradle!
[ant:echo] Aber ich hab den Ant in mir!
```

# 3. Ring frei!


Dependency-
Management
Performance
Multi-
Module-
Support
Lizenz
IDE-Support
Support

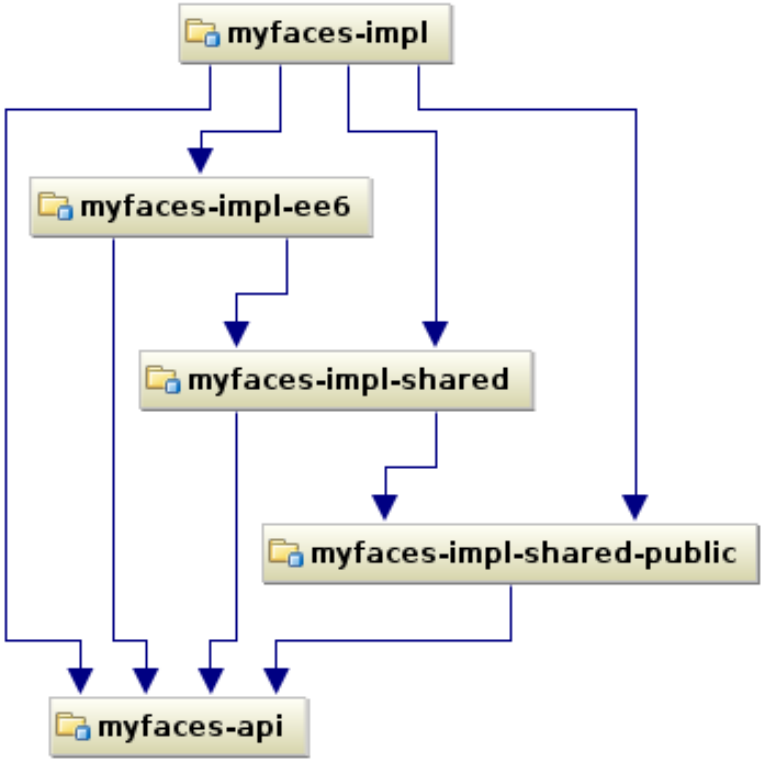
# Dependency-Management

| | Maven | Ivy | gradle |
|---|---|---|---|
| Classpathkonfiguration | Fest vorgegeben | Einstellbar | Einstellbar (+Plugins) |
| Mapping transitiver Abhängigkeiten | Automatisch | Einstellbar | Einstellbar |
| Versionierungsschema | Fest vorgegeben | Konfigurierbar | Konfigurierbar |
| Bezugsquellen | Repository | flexibel | flexibel |
| Ausschluss transitiver Abhängigkeiten | Einzeln | Global | Global |
| Bezug von Maven Abhängigkeiten | Ja | Weitgehend | Weitgehend |
| Bezug von Ivy Abhängigkeiten | Experimentell | Ja | Ja |

# And the winner is...

gradle

# Multi-Module-Projekt




- Projekt besteht meist aus mehreren Modulen
  - Build-Reihenfolge muss beachtet werden
  - Dokumentation über alle Module hinweg erzeugen
- Globale Build-Einstellungen

# Multi-Module-Projekte

| | Maven | Ant + Ivy | gradle |
|---|---|---|---|
| Konfiguration | Sehr einfach | Schwierig | Sehr einfach |
| Modulübergreifende Konfiguration | Vererbung | Imports | Configuration Injection, Vererbung |
| Vererbung Dependencies | Sehr einfach | Schwierig | Sehr einfach |
| Vorgabe von Dependency-Versionen | Einfach | Schwierig | Nicht möglich |
| Flexibilität | Gering | Hoch | Hoch |

# And the winner is...

gradle

# IDE-Support

- Buildausführung ermöglichen
- Syntax Highlighting
- Classpath aus Dependency-Mangement-Konfiguration erstellen
- Code Completion

# IDE-Support

| | eclipse | NetBeans IDE | IntelliJIDEA |
|---|---|---|---|
|   | ++ | ++ | ++ |
|   | + | -- | ++ |
| maven | + | ++ | ++ |
| gradle | + | -- | + |

# And the winner is...

IntelliJIDEA

# Support

- Dokumentation
- Literatur
- Community - Hilfe von Hobbysportlern
- Kommerzieller Support - Professionelle Trainingslager

# Support

| |  |  | maven | gradle |
|---|---|---|---|---|
| API-Dokumentation | ++ | ++ | ++ | ++ |
| Userguide | o | ++ | ++ | ++ |
| Community | ++ | ++ | ++ | + |
| Literatur | ++ | o | ++ | o |
| Kommerzieller Support | ++ | + | ++ | + |

# And the winner is...

maven

# Lizenz – Wer darf was?

- Alle Kandidaten stehen unter der Apache License 2.0
- Verpflichtet im Build-Kontext zu nichts:
  - Keine Nennung der Lizenz oder des verwendeten Tools nötig.
- Wenn das Tool Teil des ausgelieferten Produkts wird, sind die Lizenzbestimmungen zu beachten.

# Lizenz

| |  |  | maven | gradle |
|---|---|---|---|---|
| Verpflichtungen nur Build | Keine | Keine | Keine | Keine |
| Verpflichtungen bei Abhängigkeit | Wenige | Wenige | Wenige | Wenige |

# And the winner is...

gradle

<APACHE ANT>

ivy

maven

# Performance

| |  | gradle | maven |
|---|---|---|---|
| Effizienz | DAG | DAG | Lifecycle |
| Parallelisierung Unittests | Forking | Forking | Forking/ Mulithreading |
| Parallelisierung Multimoduleprojekte | Nein | Nein | Seit 3.0 |
| Parallelisierung innerhalb eines Targets/Tasks/Plugin-Goals | Ja | Ja | Ja |

# And the winner is...

## Der kompetente Buildmeister!

# Migration

- Ant & Ivy → Gradle
  - Ant-Targets und Macrodefs können wiederverwendet werden
  - Ivy-Konfiguration muss angepasst werden
- Maven → Gradle
  - Java-Plugin hat dasselbe Projektlayout
  - Mvn2gradle (Migrationskript)
  - Gradle-Plugin (pom.xml bleibt erhalten)
- Ant & Ivy/Gradle → Maven
  - Anpassung an Maven-Standards (Layout, Lifecycle, Plugins)
  - Überführung von Ant-Skripten in Plugins
  - Abhängigkeitsdefinitionen müssen angepasst werden

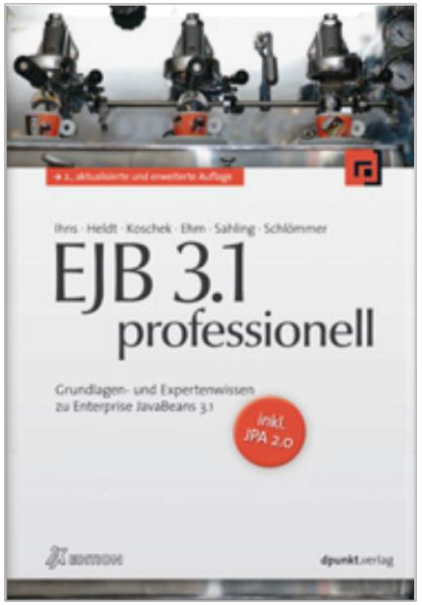
# Erwägungen bei der Toolauswahl

- Eigene Kompetenz
  - In der Builddomäne
  - Im Buildtool
- Toolsupport
- Disziplin
- Unterstützung durch Plugins bzw. Tasks
- Migrationsaufwände
- Akzeptanz durch die Beteiligten

# Wir sind...

- Holisticon AG - Management- und IT-Beratung
  - Architektur
  - Agil/Projektmanagement
  - BPM/SOA
- Mehr gibt‘s hier
  - blog.holisticon.de
  - sowie in Papierform





# Vielen Dank!

**Sven Bunge**
sven.bunge@holisticon.de

**Carl Düvel**
carl.duevel@holisticon.de