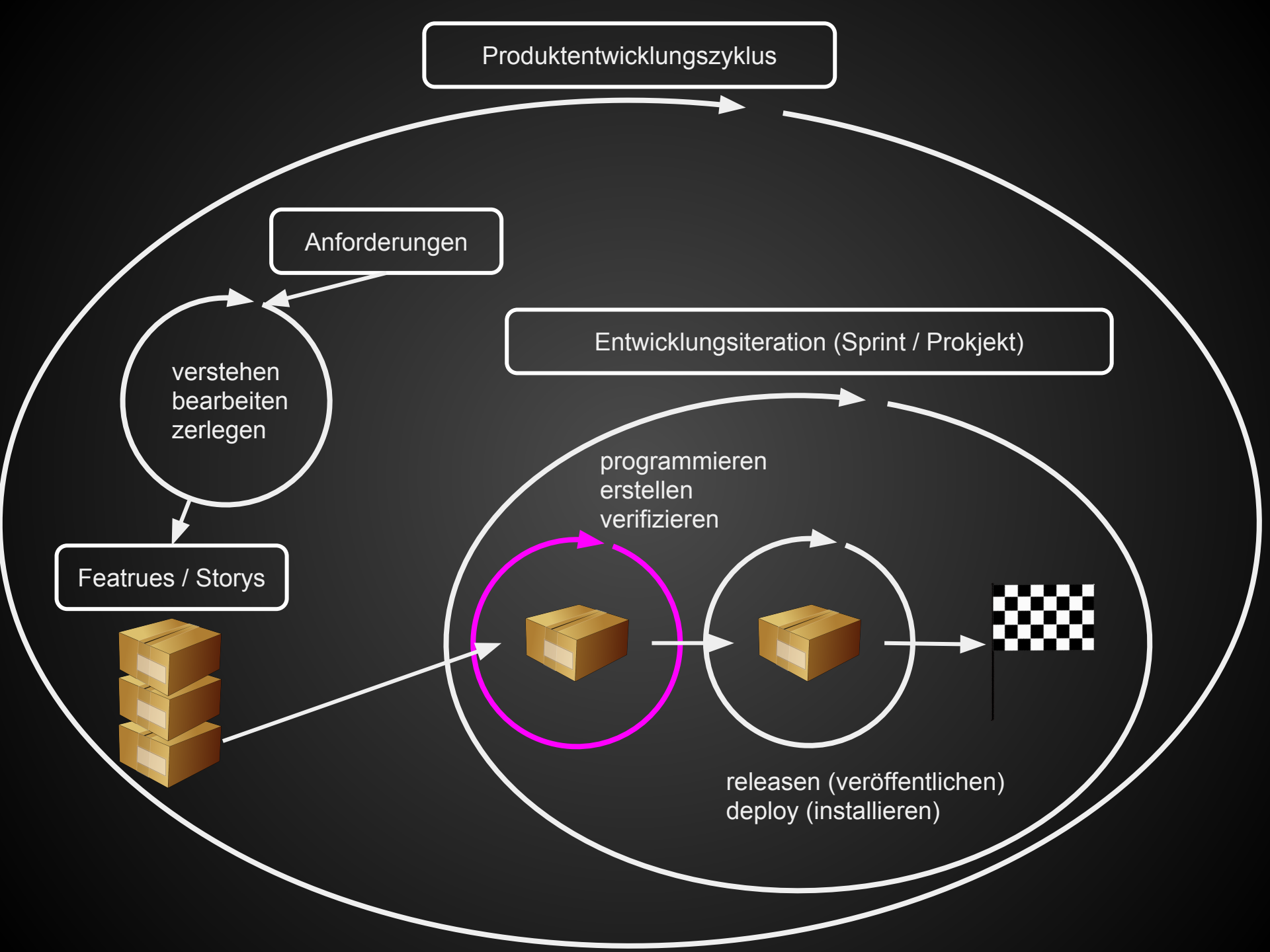
Build Tools

Gradle / Maven / Imperativ vs. Deklartiv


Produktentwicklungszyklus
Anforderungen
verstehen
bearbeiten
zerlegen
Featrues / Storys
Entwicklungsiteration (Sprint / Prokjekt)
programmieren
erstellen
verifizieren
releasen (veröffentlichen)
deploy (installieren)

Erinnerung - Was wollte ich von Ihnen?

- Software lässt jederzeit auf jeder Maschine bauen lassen
- Abhängigkeiten werden durch das automatisch aufgelöst
- Quelltext wird übersetzt
- Resourcen werden durch das BuildTool kopiert (keine manuellen Eingriffe nötig!)

Build Tools - Übersicht

- **Maven (Java/Groov)**
- **Gradle (Java/Groovy)**
- Ant (Java/Groovy)
- Grunt (JavaScript)
- Make (C/C++)
- Rake (Ruby)
- SBT (Scala)
- Leiningen (Clojure)

Deklarativ vs. Imperativ

oder konfigurieren vs. skripten
oder “Convention over Configuration”

Imperativ

“... Folge von Anweisungen (...), die vorgeben, in welcher Reihenfolge was vom Computer getan werden soll ...”

-> Programmieren mit Befehlen

z. B. Assembler, C, Java, etc.

Quelle: http://de.wikipedia.org/wiki/Imperative_Programmierung

Deklarativ

“... Paradigma bei dem die Beschreibung des Problems im Vordergrund steht. Der Lösungsweg wird dann automatisch ermittelt. ...”

-> Programmierung durch Problembeschreibung

z. B. Prolog, LISP, Erlang, SQL

Quelle: http://de.wikipedia.org/wiki/Deklarative_Programmierung

Beispielhafte Einordnung der Tools

Imperativ: Ant, Shell-Skripte

Deklarativ: Maven

Beides: Gradle, Grails

Quelle: http://maven.apache.org/

Maven

- Apache-Projekt: http://maven.apache.org
- Lizenz: The Apache Software License, Version 2.0
- erste Beta 2002
- Erste veröfentlichte Version vom 13.07.2004 -> 9 Jahre alt
- Aktuell: Maven 3.1.0 vom 15.07.2013
- XML als Build-Beschreibungssprache

Lifecycle

- klare Definition eines Prozesses um ein Artefakt (Projekt) zu bauen und zu verteilen
- Ziel: wenige Befehle sind zu lernen um jedes (!) Maven-Projekt bauen zu können (Konvention)
- 3 eingebaute “Build lifecycle”
  - default (Erstellung)
  - clean (Aufräumen)
  - site (Dokumentation erstellen)

Der “Default”-Zyklus

- **validate** - Projekte ist gültig
- **compile** - Kompilieren des Quelltextes
- **test** - Testen des übersetzen Quelltextes
- **package** - Paketieren des Kompilates
- **integration-test** - “deploy” + Integrationstests
- **verify** - alle Überprüfungen um Qualität sicherzustellen
- **install** - Paket ins lokale Repository einspielen
- **deploy** - lädt Paket in “Remote-Repository”

Ein Kommando - alle Phasen:

```
$ mvn deploy
```

weitere Befehle:

$ mvn clean

$ mvn validate

$ mvn compile

$ mvn test

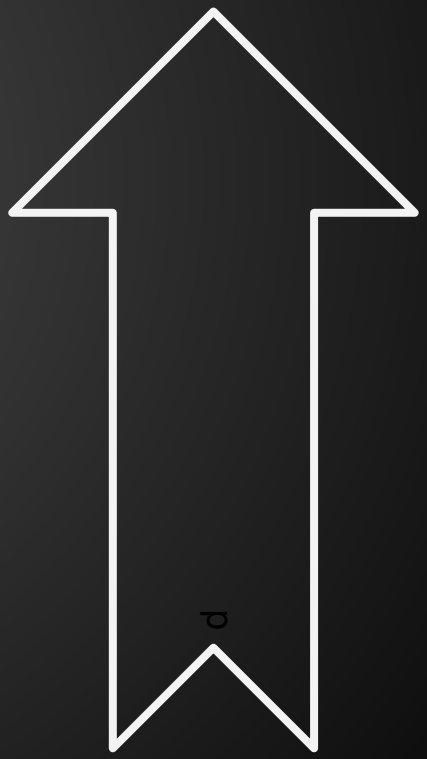
$ mvn package

$ mvn integration-test

$ mvn verify

$ mvn install

$ mvn deploy



Meine Lieblinge:

```
$ mvn clean verify
$ mvn clean install
```

pom.xml - Beispiel

```
<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>fhb</groupId>
  <artifactId>example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

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

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

  <dependencies/>

  <build/>
</project>
```

Dependencies

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-io</artifactId>
    <version>1.3.2</version>
</dependency>
```

Quelle: http://www.gradle.org/

Gradle

- Gradleware: http://www.gradle.org
- Lizenz: The Apache Software License, Version 2.0
- Open Source ähnlichen Model wie Spring Source (Geld durch Beratung)
- Aktuelle Version: Gradle 1.7 vom 06.08.2013
- Groovy als Build-Beschreibungssprache (DSL)

Lifecycle

- vereinigt “build-by-convention” mit Skript-Anteilen wie bei Ant
- Task anstelle von “Life Cycle”-Phasen
- 3 eingebaute “Build”-Phasen
  - initalization (Relevant für Multi-Projekte)
  - configuration (Aufbau DAG - directed acyclic graph)
  - execution (Ausführung der Task entsprechend DAG)

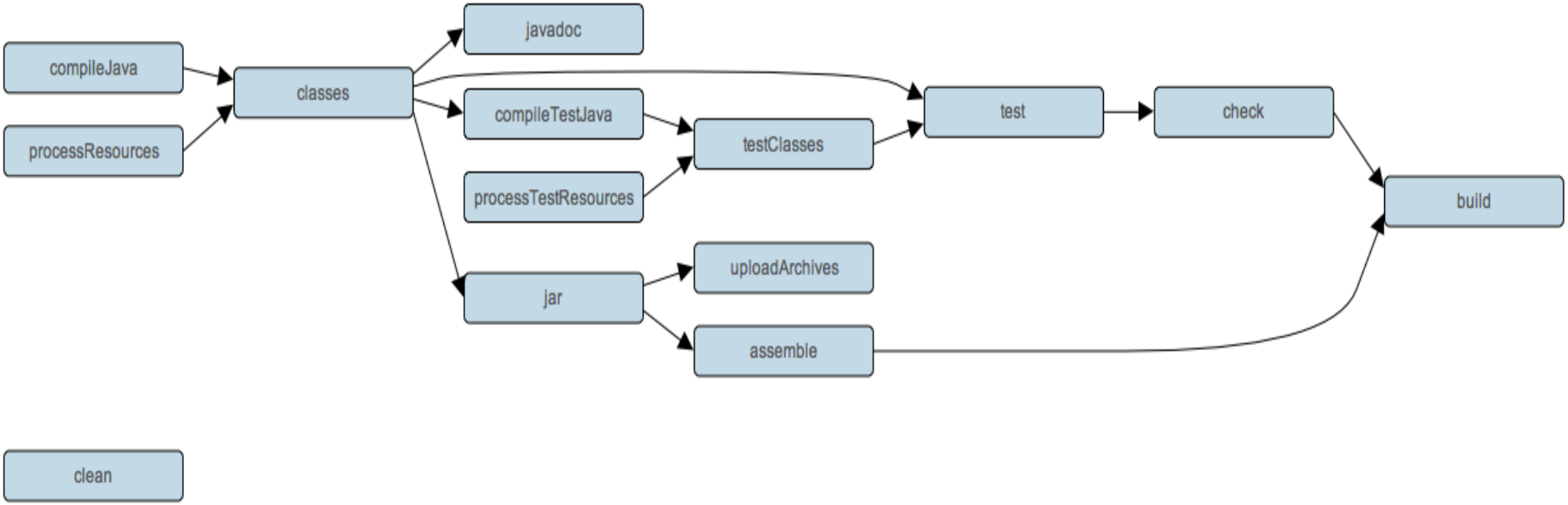
Tasks statt Phasen

- wie bei Ant werden Task ausgeführt
- Plugins definieren eine Standard “Task”-Reihenfolge
- Task-Reihenfolge kann beeinflusst werden
- DAG - Gerichteter zyklenfreier Graph bestimmt Ausführungsreihenfolge

Java Plugin - Task und Reihenfolge

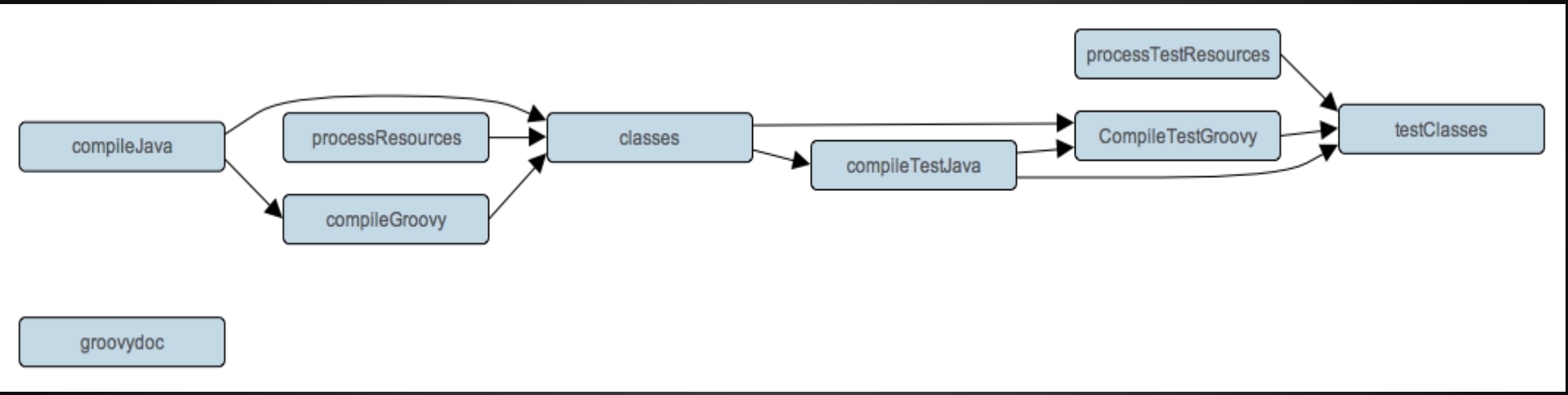
:compileJava
:processResources
:classes
:jar
:assemble
:compileTestJava
:processTestResources
:testClasses
:test
:check
:build

Java-Plugin - DAG



Quelle: http://www.gradle.org/docs/current/userguide/java_plugin.html

Groovy Plugin - DAG



Quelle: http://www.gradle.org/docs/current/userguide/groovy_plugin.html

Ein Kommando - alle Tasks:

```
$ gradle build
```

Die wichtigsten (Lifecycle)-Tasks

$ gradle clean - (aufräumen)
$ gradle assemble - (alle Archive)
$ gradle check - (alle Checks)
$ gradle build - (alle Tasks)

build.gradle

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

group="de.fhb"
version="1.0-SNAPSHOT"
description = "Example Java Project with IDE integration"

repositories {
    mavenCentral()
}

dependencies {
}
```

Dependecies

```
testCompile 'junit:junit:4.11'

testCompile  group: 'junit', name: 'junit', version:
                          '4.11'
```

Fragen?