GRUNDLEGENDE TOOLS DES JAVA ENTWICKLERS

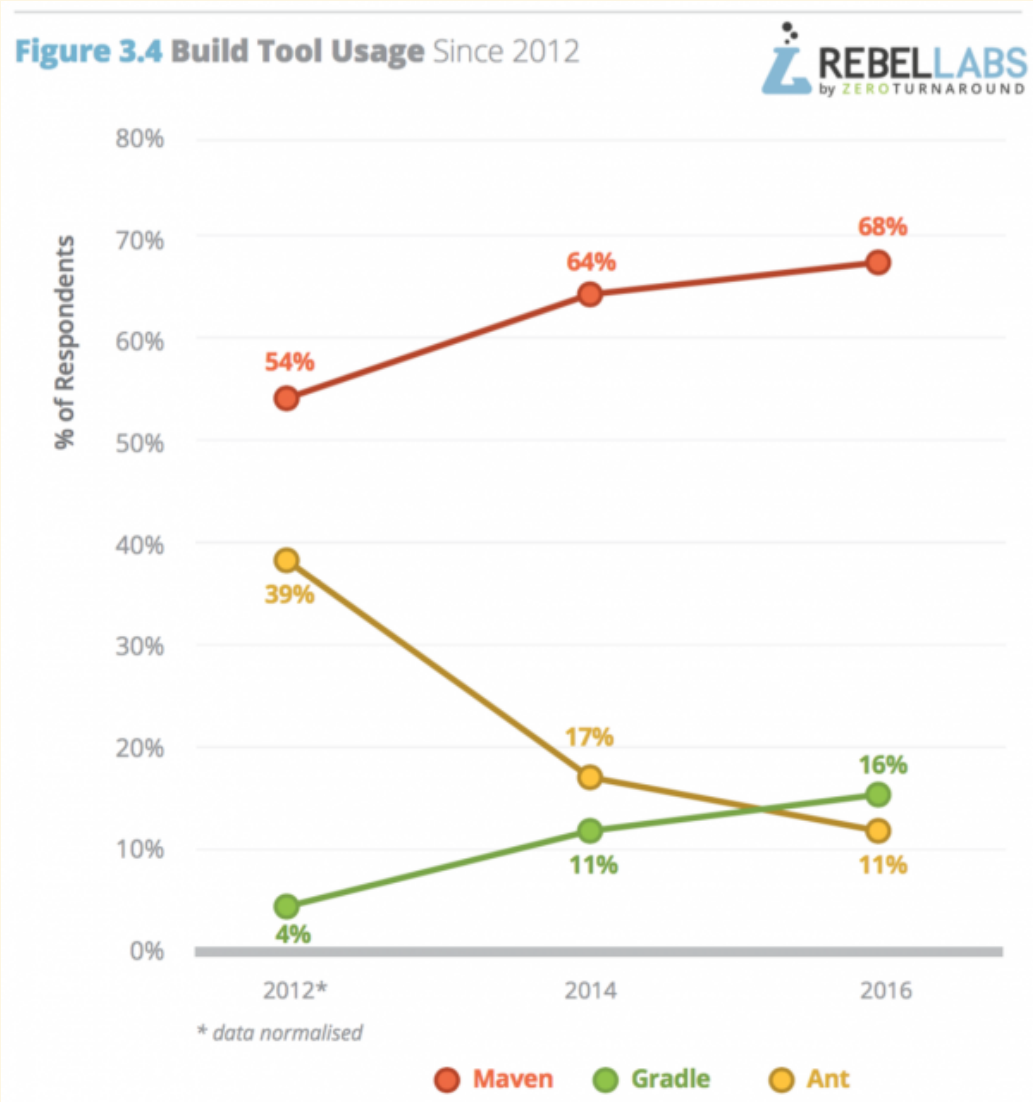
INHALTE

1. Der Software Entwicklungsprozess
2. Maven Grundlegendes
3. IDE's
4. Maven Plugins
5. Arbeitsweisen

DER SOFTWARE ENTWICKLUNGSPROZESS

SDLC - SOFTWARE DEVELOPMENT LIFE CYCLE

JAVA BUILDMANAGEMENT TOOLS



Veröffentlicht	Tool	Market Share	Open Source	Based
2000	ant	11%	✓	xml
2004	maven	68 %	✓	xml
2012	gradle	16 %	✓	DSL

MAVEN

Auf Java basierendes Buildmanagement Tool. Zur Konfiguration wird die **pom.xml** verwendet. In ihr wird das jeweilige Projekt beschrieben.

Es ermöglicht folgende Punkte:

- Kontrolle von Abhängigkeiten
- Pflege von Metadaten (Lizenz Informationen, Entwickler, Webseite...)
- Orchestrierung
- Versionierung
- Automatisierung
- Integriertation von Tools

MAVEN - KOORDINATEN

Mit "Koordinaten" werden die fünf Informationen bezeichnet mit dennen man ein Artefakt identifizieren kann.

Eintrag	Benötigt	Beschreibung
groupId	✓	Identifiziert Firma oder Bereich aus dem eine Komponente kommt
artifactId	✓	Identifiziert das Paket
version	✓	Version des Artefakts, i.d.R.: Major.Minor.Patch
type	✗	Welches packaging wurde verwendet, z.B. jar oder war
classifier	✗	Multiple Artefakte je Pom erstellen, z.B. JDK14 / JDK15 Builds

MAVEN - ABHÄNGIGKEITEN

Die eigenen Koordinaten werden wie folgt in der pom.xml innerhalb des `project` Knotens definiert.

```
<project>
...
  <groupId>com.dhbw.mdse</groupId>
  <artifactId>ugly_project_step1</artifactId>
  <version>1.0</version>
...
</project>
```

MAVEN - ABHÄNGIGKEITEN

Java Anwendungen haben in der Regel eine vielzahl von Abhängigkeiten. Diese werden in der dependency Sektion angegeben.

```
<project>
...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
...
</project>
```

MAVEN - POM.XML

Eintrag	Benötigt	Beschreibung
modelVersion	✓	Version der verwendeten Strukturdefinition, z.B. 4.0.0
packaging	✓	Typ des Artefaktes, z.B. `POM`, `EAR`, `WAR` oder `JAR`, Default: jar
properties	✗	Definition und Nutzung von variablen Einstellungen
dependencies	✗	Listet benötigte Abhängigkeiten auf
build	✗	Build Informationen
profiles	✗	Definition von unterschiedliche Verhalten, z.B. zur CodeAnalyse
reporting	✗	Erstellung von Reports

MAVEN - LIFECYCLE

Phase	Beschreibung
validate	Prüfung, dass notwendige Informationen vorhanden sind
compile	Umwandeln des Projekts in Bytecode
test	Ausführen von Unit Tests
package	Typ des Artefaktes, z.B. `POM`, `EAR`, `WAR` oder `JAR`, Default: jar
verify	Ausführung von Integrations-Tests
install	Installieren des Artefakts im lokalen maven Repository (.m2)
deploy	Liefert das Artefakt in ein zentrales maven Repository um es mit anderen Entwicklern zu teilen

MAVEN - LIFECYCLE

Es gibt noch weitere default Lifecycle, z.b. für Site. Diese sind auf https://maven.apache.org zu finden.

MAVEN CLI

```
mvn clean
mvn compile
mvn test
mvn package
mvn package -DskipTests
mvn install
```

MAVEN

- **Goals:**

  Einzelne Komponenten können Funktionen exponieren. Diese nennt man Goals. Goals können an Phasen gebunden werden, aber auch einzeln ausgeführt werden. Darüber können sich Maven Plugins am Lifecycle beteiligen.

- **Archetypes:**

  Templates mit dennen man Projekte ausliefern kann.

ÜBUNG

MAVEN IN FIVE MINUTES

https://tiny.cc/m5minutes

IDE'S

Integrierte Entwicklungsumgebungen integrieren Funktionen zur Software Entwicklung:

- Editor
- Syntax Highlighting
- Code Generierung
- Automatisierte Fehlerbehandlung
- Integration von Tools
- Debugging
- Versionskontrolle
- und und und...

IDE'S

INTELLIJ IDEA

MAVEN PLUGINS

Es gibt eine Reihe von Plugins die man in Maven nutzen kann. Sie decken folgenden Funktionen ab:

- Validierung
  - Code Format (Checkstyle, ArchUnit)
  - Common Vulnerabilities and Exposures(CVE) Scans
  - Code Qualität (PMD, SpotBugs, SonarQube)
  - Testing (Surefire)
- Reporting (JavaDocs, pdf)
- Assembly

MAVEN PLUGINS

Bezeichnung	Funktion
Surefire	Stellt Ergebnisse der Tests als Report zur Verfügung
JavaDocs	Erstellt Dokumentation auf Basis der Struktur der Codes und von Kommentaren die Entwickler schreiben
Checkstyle	Überprüft die Formatierung des Codes
SpotBugs	Statische Code Analyse
PMD	Statische Code Analyse
SonarQube	Statische Code Analyse

JAVADOCS

Kommentarblöcke die mit "**" beginnen, werden als JavaDocs interpretiert. Diese können an Packages, Klassen, Interfaces, Methoden, Feldern und Variablen genutzt werden.

```
/**
* Adds the two provided values.
*
* @param var1 first input value
* @param var2 second input value
*/
 public final void add(final int var1, final int var2) {
```

JAVADOCS

Annotation	Beschreibung
@author	Nennt den Author einer Klasse
@version	Gibt die Version der Klasse an
@deprecated	Gibt an das eine Methode nicht mehr zu verwenden ist
@param	Beschreibt die Parameter einer Methode
@return	Beschreibt den Rückgabewert einer Methode
@throws	Beschreibt die Fehler welche von einer Methode ausgelöst werden können

CHECKSTYLE / CODE STYLE

- **Checkstyle (maven)** überprüft das Format des Codings. Dies kann durch eine xml Datei angepasst werden.
- **Code style (IDE)** formatiert den Code.

=> Wichtig das beide Tools möglichst synchron sind.

UNIT TESTING - JUNIT

- Testet kleine funktionelle Einheiten, möglichst Methoden
- Test wird auch Unit Test genannt
- Test in sich geschlossen.
- Wird in der Test Phase ausgeführt
- Von Entwicklern, für Entwickler

JUNIT

Annotation	Wo	Wofür
@BeforeClass	Methode	Einmalige Ausführung der Methode zum Setup der Klasse
@AfterClass	Methode	Einmalige Ausführung der Methode zum Cleanup der Klasse
@Before	Methode	Ausführung der Methode zum Setup je Test in Klasse
@After	Methode	Ausführung der Methode zum Cleanup je Test in Klasse
@Test	Methode	Methode wird als Test deklariert

ÜBUNG 2

ARBEITSWEISEN

- Test Driven Development
- Pair Programming
- Mob Programming

TDD - TEST DRIVEN DEVELOPMENT

1. Test wird entwickelt, schlägt aber fehl.
2. Test wird mit minimalen Änderungen Grün gemacht. Copy Paste, oder ähnliches ist erlaubt und gewünscht, hauptsache der Test ist mit wenig Aufwand wieder grün.
3. Refaktorierung sodass auch die Qualität, Struktur und Architektur in Ordnung ist

PAIR PROGRAMMING

- 2 Entwickler aber nur 1 Computer
- Navigator sagt was gemacht wird
- Driver bedient Computer
- Rollen werden regelmäßig getauscht
- Besonders geeignet bei
  - unterschiedlichen Skill Level
  - Anforderung an besonders hohe Software Qualität

MOB PROGRAMMING

- 3-5 Entwickler aber nur 1 Computer
- Es gibt einen Driver, der rotiert
- Es muss Einigkeit über das herrschen was getan werden soll
- Geteiltes Wissen im Team (Code Ownership)
- Zeitlich beschränkt, mit regelmäßigen Pausen (ca. alle 45 min)
- Keine Handys oder ähnliche Ablenkungen

END