Reporting & Dokumentation

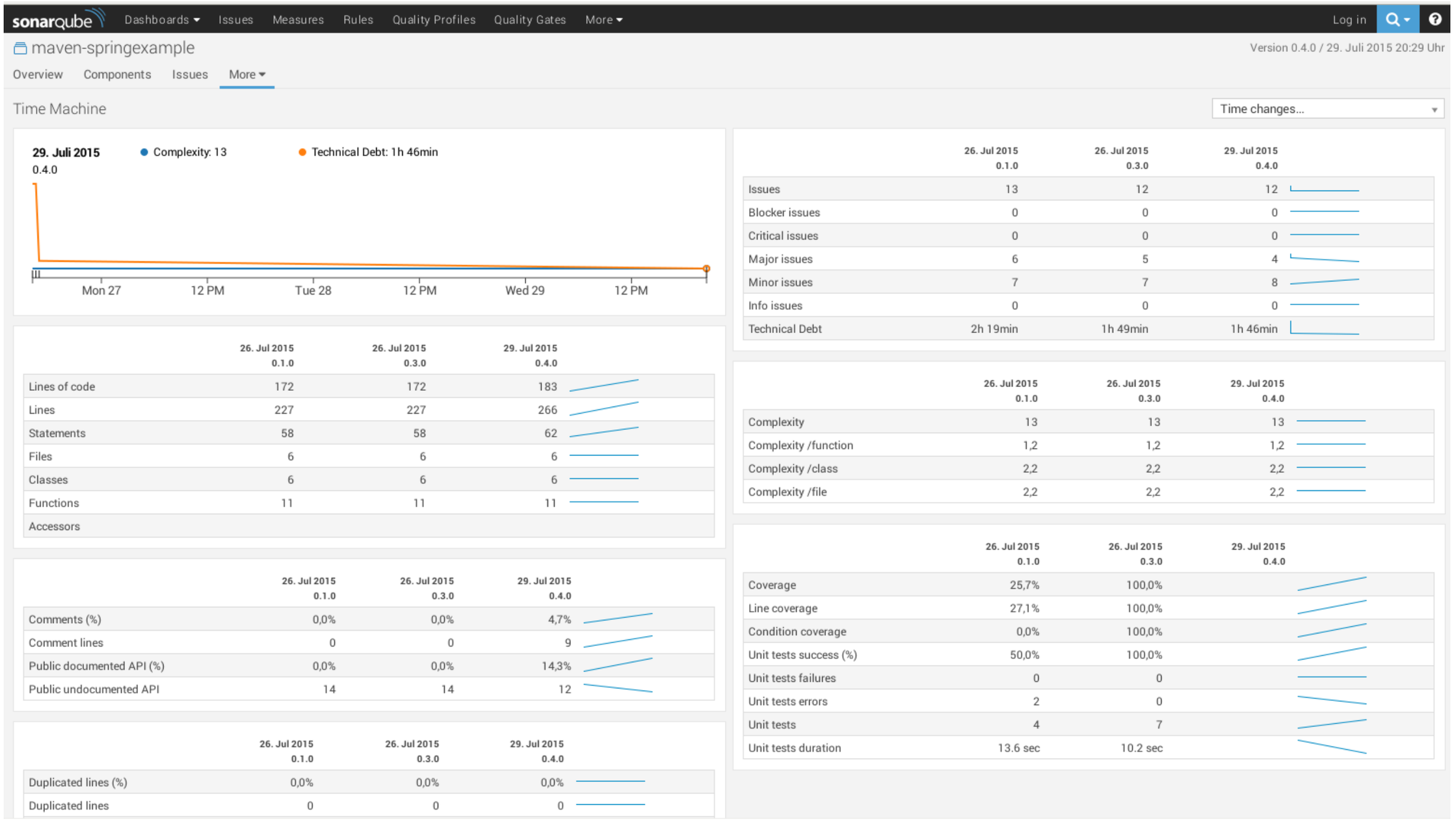
Worum geht es?

- Im Rahmen des Builds können automatisiert Dokumente zum Projekt erzeugt werden
- Reports zur Codequalität
  - Erstellung von Diagrammen, die Qualitätsmetriken (über die Zeit) visualisieren
- Diagramme zu Datenbankschemas
- UML-Diagramme
- Reports mit Informationen aus Drittsystemen (z.B. Bugtracker, Versionsverwaltung)
- Dokumente aus Kommentaren im Quelltext

Codequalität - Erstellung von Reports

- IDE
  - Plugins PMD, EclEmma (JaCoCo), Findbugs, Checkstyle
- Maven
  - Reporterstellung dauert relativ lange
    - Schnelle Ausführung und Rückmeldung im Build erwünscht
  - Vorteil: Ist unabhängig von SonarQube
  - Teilweise Integration in Jenkins
- SonarQube
  - Umfangreiche Analysen und Historie
  - "Sonar Way" vs. PMD/Findbugs/Checkstyle (deprecated)

SonarQube


sonarqube
Dashboards
Issues
Measures
Rules
Quality Profiles
Quality Gates
More
Log in
maven-springexample
Version 0.4.0 / 29. Juli 2015 20:29 Uhr
Overview
Components
Issues
More
Time Machine
Time changes...
29. Juli 2015
0.4.0
Complexity: 13
Technical Debt: 1h 46min
Mon 27
12 PM
Tue 28
12 PM
Wed 29
12 PM
26. Jul 2015 0.1.0
26. Jul 2015 0.3.0
29. Jul 2015 0.4.0
Issues 13 12 12
Blocker issues 0 0 0
Critical issues 0 0 0
Major issues 6 5 4
Minor issues 7 7 8
Info issues 0 0 0
Technical Debt 2h 19min 1h 49min 1h 46min
Lines of code 172 172 183
Lines 227 227 266
Statements 58 58 62
Files 6 6 6
Classes 6 6 6
Functions 11 11 11
Accessors
Complexity 13 13 13
Complexity /function 1,2 1,2 1,2
Complexity /class 2,2 2,2 2,2
Complexity /file 2,2 2,2 2,2
Comments (%) 0,0% 0,0% 4,7%
Comment lines 0 0 9
Public documented API (%) 0,0% 0,0% 14,3%
Public undocumented API 14 14 12
Coverage 25,7% 100,0%
Line coverage 27,1% 100,0%
Condition coverage 0,0% 100,0%
Unit tests success (%) 50,0% 100,0%
Unit tests failures 0 0
Unit tests errors 2 0
Unit tests 4 7
Unit tests duration 13.6 sec 10.2 sec
Duplicated lines (%) 0,0% 0,0% 0,0%
Duplicated lines 0 0 0

Empfehlung Codequalität

- Standardregeln vs. angepasste Regeln
- Performance - Abwägung zwischen schnellem Build
- meist nur bei neuen Projekten möglich, alte Projekt haben oft "Debts"
- Ist nicht der "heilige Gral"
  - Nicht alle Metriken sind sinnvoll
  - False Positives
  - Aber: Code sauber halten

Javadoc

- Dokumentationswerkzeug, des JDK
  - generiert aus Java-Quellcode HTML-Dateien
  - enthalten sind alle Kommentare die einer bestimmten Struktur entsprechen
  - zur Strukturierung der Dokumentation existieren eigene Annotations
- wird hauptsächlich für API-Dokumentation verwendet
- Konfiguration mittels maven-javadoc-plugin
  - https://maven.apache.org/plugins/maven-javadoc-plugin/

Beispiel generierte Doku

Constructor Summary

Constructors

Constructor and Description
`ArrayList()` Constructs an empty list with an initial capacity of ten.
`ArrayList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayList(int initialCapacity)` Constructs an empty list with the specified initial capacity.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
boolean	`add(E e)` Appends the specified element to the end of this list.
void	`add(int index, E element)` Inserts the specified element at the specified position in this list.
boolean	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	`clear()` Removes all of the elements from this list.
`Object`	`clone()` Returns a shallow copy of this `ArrayList` instance.
boolean	`contains(Object o)` Returns `true` if this list contains the specified element.
void	`ensureCapacity(int minCapacity)` Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the

Beispiel Java-Quellcode

```java
/**
 * Constructs an empty list with the specified initial capacity.
 *
 * @param  initialCapacity  the initial capacity of the list
 * @throws IllegalArgumentException if the specified initial capacity
 *         is negative
 */
public ArrayList(int initialCapacity) {
    super();
    if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal Capacity: "+
                                           initialCapacity);
    this.elementData = new Object[initialCapacity];
}

/**
 * Constructs an empty list with an initial capacity of ten.
 */
public ArrayList() {
    super();
    this.elementData = EMPTY_ELEMENTDATA;
}

/**
 * Constructs a list containing the elements of the specified
 * collection, in the order they are returned by the collection's
 * iterator.
 *
 * @param c the collection whose elements are to be placed into this list
 * @throws NullPointerException if the specified collection is null
 */
public ArrayList(Collection<? extends E> c) {
    elementData = c.toArray();
    size = elementData.length;
    // c.toArray might (incorrectly) not return Object[] (see 6260652)
    if (elementData.getClass() != Object[].class)
        elementData = Arrays.copyOf(elementData, size, Object[].class);
}

/**
 * Trims the capacity of this <tt>ArrayList</tt> instance to be the
 * list's current size.  An application can use this operation to minimize
 * the storage of an <tt>ArrayList</tt> instance.
 */
public void trimToSize() {
```

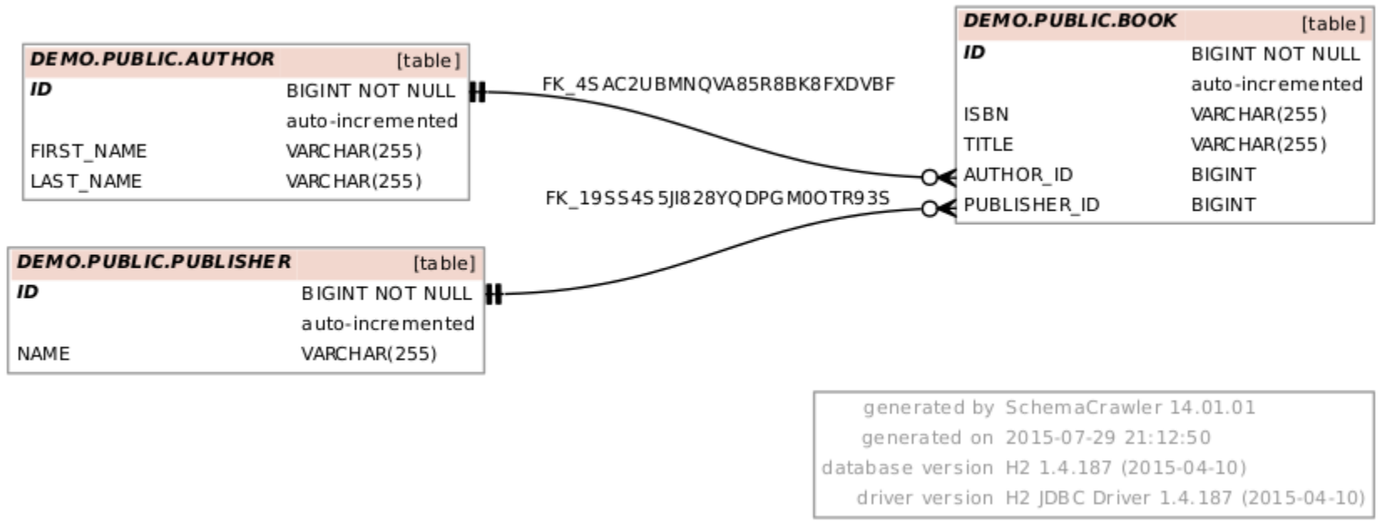
Beispiel Maven Konfiguration

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>2.10.3</version>
    <configuration>
      ...
    </configuration>
</plugin>
```

Datenbankdiagramm generieren

- Generieren eines ER-Diagramms zu einer bestehenden Datenbank
- Beispielsweise mittels SchemaCrawler (http://sualeh.github.io/SchemaCrawler/)
- Dafür existiert kein fertiges Maven Plugin
- Konfiguration mittels `exec-maven-plugin`

Datenbankdiagramm



mvn site

- Generiert eine HTML Seite aus Informationen in der pom.xml
- Ausserhalb von OpenSource Projekten praktisch keine Verwendung

Übung 5

- Wechsle in das Verzeichnis `if2015-maven` und führe dort `git pull` aus
- Die Sourcen für diese Übung befinden sich im Verzeichnis `uebung-spring3`
- Füge das Repository
  `http://52.18.220.227:8081/nexus/content/repositories/releases/`
  zum Downloaden von dependencies hinzu
- Ergänze die Dependency
  `de.informatica.buildmanagement:documentation:0.5`
- Konfiguriere das JavaDoc Plugin mit dem Custom Doclet `doclets.GlossarDoclet`. Dieses erzeugt ein einfaches Glossar für das Projekt. Setze dabei auch den Konfigurationsparameter
  `<useStandardDocletOptions>false</useStandardDocletOptions>`
- Generiere mit `mvn javadoc:javadoc` ein Glossar

- Lade SonarQube als ZipFile herunter (http://www.sonarqube.org/downloads/), entpacke und starte es
- Erzeuge mittels `mvn clean install sonar:sonar` einen neuen Report
- Nimm einige Änderungen am Quelltext vor und beobachte wie sich die Metriken verändern

Versionierung & Releases

Versionierung - The Maven Way

- Während der Entwicklung wird an die Versionsnummer `-SNAPSHOT` angehängt
  - z.B. `<version>0.6-SNAPSHOT</version>`
- Ist die Entwicklung einer Version "fertig", wird ein Release erstellt
  - z.B. `<version>0.6</version>`
- Wird die Entwicklung fortgesetzt, wird die Versionsnummer hochgezählt und wieder `-SNAPSHOT` angehängt
  - z.B. `<version>0.7-SNAPSHOT</version>`

Erstellen von Releases

- Philosophie: Man erstellt ein Release, wenn ein Snapshot-Build gut genug ist
- Maven Release Plugin
  - http://maven.apache.org/maven-release/maven-release-plugin/
- Was macht das Maven Release Plugin?
  - Release Version setzen, neues Tag in der Versionsverwaltung erstellen
  - Bauen des Projektes
  - neue Snapshot Version setzen
  - Deployment auf Archivserver
- Arbeitet auf master, während des Releases (prepare) darf kein Commit erfolgen

Konfiguration des Maven Release Plugins

Archivserver

```
<distributionManagement>
  <repository>
    <id>informatica</id>
    <url>http://52.18.220.227:8081/nexus/content/repositories/releases/</url>
  </repository>
  <snapshotRepository>
    <id>informatica</id>
    <url>http://52.18.220.227:8081/nexus/content/repositories/snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

Repository der Versionsverwaltung

```
<scm>
  <developerConnection>scm:git:https://github.com/cko/myproject.git</developerConnection>
  <tag>HEAD</tag>
</scm>
```

Release Plugin hinzufügen

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-release-plugin</artifactId>
  <version>2.5.2</version>
</plugin>
```

Release erstellen

Zwei Schritte - prepare & perform

```
mvn release:prepare

mvn release:prepare -DdryRun=true

mvn release: perform

mvn release:prepare release:perform
```

Aufräumen eines fehlgeschlagenen Releases

```
mvn release:clean
```

Versionierung - The Continuous Delivery Way

- Philosophie: Jeder Build führt zu einem potentielles Release
- Forlaufende Nummerierung der Build Artifakte bei jedem Build
- Passt nicht zum Release Plugin

Konfigurationsbeispiel Continuous Delivery

- Entwicklung (master) hat die Version `1.0-SNAPSHOT`
- Commit triggert Build
  - Jenkins ermittelt Build Nummer
  - `git checkout -b 1.0.$BUILD_NUMBER`
  - `mvn versions:set -DnewVersion=1.0.$BUILD_NUMBER`
  - `mvn clean install`
  - Bei Erfolg:
    - `git commit && git push`
    - `mvn deploy`

Versions Plugin

- Setzen der Projektversion: `mvn versions:set -DnewVersion=1.2.3`
- Anzeige von neueren Dependency und Plugin Versionen
- Setzen von neueren Dependency Versionen

http://www.mojohaus.org/versions-maven-plugin/

Übung 6

- (Lege einen GitHub Account an)
- Erstelle ein neues Git Repository auf GitHub
- Pushe Dein Projekt aus Übung 5
- Erstelle mit dem Maven Release Plugin ein Release

Parent POM

Parent POM

- Mehrere Projekte teilen sich eine POM, die gemeinsam genutzte Elemente definiert
  - Gemeinsame Abhängigkeiten und DependencyManagement
  - Properties
  - Plugin Konfigurationen
  - Angaben zur Entwicklungsinfrastruktur: Git Repositories, Archivserver/Repositories

Verwendung einer Parent POM

1. Parent POM anlegen und deployen
2. Parent POM ins Projekt einbinden

Parent POM anlegen

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.informatica-feminale</groupId>
  <artifactId>buildmanagement-parent</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
  ...
</project>
```

Deployen einer Parent POM

Mit dem Releas Plugin

```
mvn -N -Darguments=-N release:prepare
mvn -N -Darguments=-N release:perform
```

Alternativ

```
mvn -N deploy
```

Achtung: Versionsnummer!

Parent POM in die Projekt POM einbinden

```
<project>
    <parent>
        <groupId>de.informatica-feminale</groupId>
        <artifactId>buildmanagement-parent</artifactId>
        <version>1.0</version>
    </parent>
    ...
</project>
```

Effektive POM

- Die eigene POM und (rekursiv alle) Parent POMs werden zusammengeführt
- Anzeigen mit

```
mvn help:effective-pom
```

Profile

Zweck von Profilen

- Mit Profilen können Teile des Builds bedingt ausgeführt werden
- Steuerung über Kommandozeilenparameter
- Beispiele
  - Langlaufende Test
  - Erweiterte Checks oder GPG Signierung
  - Datenbank für Tests

Konfiguration von Profilen

```
<profiles>
    <profile>
      <id>test</id>
      <activation>...</activation>
      <build>...</build>
      <repositories>...</repositories>
      <pluginRepositories>...</pluginRepositories>
      <dependencies>...</dependencies>
      <dependencyManagement>...</dependencyManagement>
      <distributionManagement>...</distributionManagement>
    </profile>
</profiles>
```

Konfiguration

Profile können auf zwei verschiedene Arten konfiguriert werden:

- POM.xml
- settings.xml
  - sowohl in der Maven Installation, als auch in den User settings

Aktivierung von Profilen

- In der Profil Konfiguration können Aktivierungsbedingungen konfiguriert werden
  - `<os>`: Wenn der Build auf einem bestimmten Betriebssystem läuft
  - `<jdk>`: wenn eine bestimmte Java Version vorhanden ist
  - `<file>`: Wenn eine Datei existiert `<exists>` oder fehlt `<missing>`
  - `<property>`: Wenn ein Systemproperty oder Umgebungsvariable `env.FOO` auf einen bestimmten Wert gesetzt ist
- Aktiviertung "by Default"
- Explizit bei Aufruf von Maven mit dem `-P` Parameter, z.B. `-Pprofile-id`

Anzeige der aktiven Profile mit `mvn help:active-profiles`

Übung 7

- Erstelle eine Parent POM für das Projekt aus Übung 5/6
- Verwende dafür die groupId `de.informatica-feminale` und die artifactId `buildmanagement-parent-<vorname>`.
- Verschiebe dorthin die Property für die Spring Version und die Konfiguration des `distributionManagement`.
- Lege für die Parent POM ein neues GitHub Repository an
- Deploye die Parent POM auf den Nexus
- Binde die Parent POM in dein Projekt ein und pushe die Änderungen zu GitHub