MOTIVATION AUTOMATISCHER BUILD

In der Softwareentwicklung gibt es zahlreiche Arbeitsschritte, die sehr oft wiederholt werden müssen (wie z.B. Source Code kompilieren, Testfälle ausführen, Software ausliefern).

Viele dieser Arbeitsschritte werden bereits durch moderne Entwicklungsumgebungen automatisiert durchgeführt. Es gibt jedoch immer Szenarien, wo keine umfangreiche Entwicklungsumgebung wie Eclipse oder Intellij vorhanden ist. So müssen diese Arbeitsschritte entweder manuell stets in der gleichen Reihenfolge wiederholt werden oder durch einen automatischen Build Vorgang einmalig definiert und beliebig oft „per Knopfdruck" wiederholt werden.

Selbst wenn Entwicklungsumgebungen vorhanden sind, nutzen diese Tools ebenso die Mechanismen und Frameworks zum automatischen Build. Diese Übung dient daher zum Einstieg in die Grundlagen von Maven und Ant. Die Inhalte zum automatischen Build werden in den nächsten Einheiten erweitert.

ZIEL DER ÜBUNG

- Sie können mittels Maven ein Projekt builden und eine ausführbare Jar Datei erzeugen.
- Sie können manuell eine Jar Datei erzeugen.
- Sie können ein Ant-Projekt aus einer Maven-Struktur nachbauen.
- Sie können zwischen Ant-Projekte und Maven-Projekte unterscheiden.
- Sie können ein Ant build.xml Datei erstellen.
- Sie können Ant Targets erzeugen und in Abhängigkeit bringen.
- Sie können mit Ant eine ausführbare Jar Datei erzeugen.

EINLEITUNG

- Arbeiten Sie folgende Aufgabenstellung schrittweise durch. Anhand der einzelnen Schritte sollte es Ihnen möglich sein die Übung zum Automatischen Build zu absolvieren.
- Arbeiten Sie parallel mit den Folien zum Automatischen Build von Herrn Teiniker. Es wird auch auf seine Online Tutorials auf GitHub verwiesen.
- Achten Sie auf die unterschiedlichen Projektstrukturen zwischen Maven-Projekt und Ant-Projekt. Nutzen Sie dabei die Unterlagen zum Thema Projektstruktur.
- Bauen Sie den Maven Build mit Ant nach. Nutzen Sie die Ihnen zur Verfügung stehenden Unterlagen und Code Snippets.
- Nutzen Sie diese Übung um sich mit den Entwicklungssystemen Eclipse und Intellij vertraut zu machen. Führen Sie den Automatischen Build mit Ant und Maven jedoch führen Sie den Build ebenso mit dem Terminal durch. Die Übungen werden nur mit dem Terminal kontrolliert.

DAUER DER ÜBUNG

90-120 Minuten

AUFGABENSTELLUNG

MAVEN

- Verwenden Sie das Simple-Automatic-Build-Project.
- Klonen Sie folgendes Repository: https://github.com/michaelulm/AutomaticBuild.git
- Builden Sie das Projekt mit Maven und erzeugen Sie somit eine Jar Datei.

MANUELL JAR DATEI ERZEUGEN

- Bauen Sie das Maven-Projekt zu einem Ant-Projekt um. Achten Sie dabei besonders auf die Adaptierung der Projektstruktur.
- Erzeugen Sie manuell eine Jar Datei.

ANT

- Nachdem Sie manuell eine Jar Datei erzeugt haben, erstellen Sie eine build.xml Datei im Root Verzeichnis Ihres Projekts.
- Erstellen Sie auf dem FH Git-Server ein eigenes Repository https://git-iit.fh-joanneum.at/YOURNAME/ss18-configurationmanagement-exercises.git (mit den Unterordner **building/ant-simple-project = root Ordner Übung**). Erstellen Sie entweder ein public oder private (+ Zugriffsrechte für Michael Ulm) Repository.
- Clonen Sie das Repository https://git-iit.fh-joanneum.at/UlmMi/ss18-configurationmanagement.git und fügen Sie den Link zu Ihrem erstellten Repository unter building/exercises-list.txt in einer eigenen Zeile hinzu. Pro Zeile darf nur ein Link stehen.
  Hinweis: Der Link zu Ihrem Repository wird zur Beurteilung der Übung zum Automatischen Build herangezogen. Arbeiten Sie wie folgt die einzelnen Schritte durch.
- Klonen Sie Ihr Projekt in Ihre Arbeitsumgebung.
- Kopieren Sie Ihr gesamtes Projekt in das root Verzeichnis des neuen Repository (außer .git, denn damit würde Sie die Git Initialisierung des Ordners überschreiben)
- Adaptieren Sie die Maven Projektstruktur auf eine Ant-Projektstruktur.
- Versionieren Sie initial das adaptierte Projekt.
- Versionieren Sie nach jedem erstellten Target.
- Erzeugen Sie ein Target nach dem anderen und testen Sie das erstellte Target mittels

```
ant <target-name>
```

- Erweitern Sie den Output der Java Klasse um Ihr Personenkennzeichen.

- Ein **Repository** angelegt haben
- Ihr **build.xml** versioniert haben
- und die **ausführbare Jar** Datei ebenso Ihr Personenkennzeichen ausgibt
- sowie Ihre Änderungen auch mit git push in Ihr Remote Repository übertragen haben,

Erfüllen Sie die genannten Punkte, dann haben Sie diese Übung erledigt. Vergessen Sie nicht den Link zu Ihrem Repository ebenso zu veröffentlichen.

HILFESTELLUNG

MAVEN KOMMANDOS

Basierend auf den Maven Anleitungen von
https://github.com/teiniker/teiniker-lectures-configurationmanagement/tree/master/building
einige wichtige Kommandos zusammengefasst und mit weiteren Kommentaren versehen.

Die wichtigsten Maven Kommandos in der ausgeführten Reihenfolge bzw. mit deren Abhängigkeiten zu weiteren Kommandos. Maven führt in Abhängigkeit je nach Kommando die benötigten vorliegenden Kommandos ebenso in der richtigen Reihenfolge auf.

```
mvn compile
```

Es werden Abhängigkeiten zu Bibliotheken heruntergeladen und die Source Dateien kompiliert. Beim ersten Ausführen kann es dabei länger dauern, da die aktuellen genutzten Bibliotheken von Maven herunterladen werden.

```
mvn test
```

Es werden die im Testordner abgelegten Testfälle ausgeführt. Zu beachten ist die Angabe eines Testframeworks im pom.xml wie z.B. JUnit. Dabei werden die Abhängigkeiten zu Bibliotheken ebenso heruntergeladen, die Test Dateien kompiliert und ausgeführt.

```
mvn package
```

Damit wird eine Jar Datei erzeugt und im target-Verzechnis abgelegt.

JAR DATEI AUSFÜHREN

```
java -jar filename.jar Argument1 Argument2 …
```

Wenn Sie der Jar Datei eine Manifest Datei mit der Zeile „Main-Class: …“ hinzugefügt haben, so ist diese Jar Datei auch ausführbar und kann mittels java -jar … gestartet werden. Sie können Argumente der Main Methode anschließend übergeben.

ANT IM ÜBERBLICK

GRUNDLAGEN

Mit Apache Ant ist es möglich Skripte zu schreiben um bestimmte Aufgaben zu automatisieren. Darunter zählen z.B.:

- Source Code kompilieren
- Source Code Archive zu erzeugen
- Kompilierten Source Code auszuführen
- Verzeichnisse zu erzeugen
- Dokumentation zu generieren (falls diese z.B. im Source Code mitvorhanden ist)
- …

Diese einzelnen Aufgaben können entweder separat ausgeführt werden oder auch in Abhängigkeit geschalten werden. Dabei werden in einer XML Datei, meist buidl.xml benannt bzw. Build File genannt, die einzelnen Schritte definiert.

Zur Ausführung der XML Datei wird Apache Ant benötigt, das die einzelnen Targets ausliest und ausführt. Ant kann entweder über die Konsole oder über Eclipse direkt aufgerufen werden. Es empfiehlt sich in der Entwicklung das Ant Skript auch immer wieder mit der Konsole aufzurufen um mögliche Fehler erkennen zu können, da möglicherweise Eclipse und die verwendete Ant Version sich voneinander unterscheiden können.

So kann Ant entweder vom Entwickler selbst, einem Server oder von einem Entwickler auf einer Produktiven Maschine schlank und komfortabel dafür verwendet werden, um mit nur wenigen Aufrufen die gesamte Software zu generieren, sprich zu builden.

Weitere Anwendungsfälle für **Ant** sowie für **Maven** sind:

- Nightly Builds, wo jede Nacht automatisiert von einem Versionierungssystem der aktuelle Source Code Stand ausgecheckt und gebaut wird
- Continous Integration, welcher ähnlich funktioniert, jedoch zu öfteren oder anders definierten Laufzeiten, die Software gebaut und ebenfalls die Software automatisiert getestet wird, um einen aktuellen Zustand des Projekts zu ermitteln. (Lauffähig ja/nein, Testabdeckung, etc.)
- Bauen von komplexen Projektabhängigkeiten, welche durch ein Skript wesentlich komfortabler funktioniert als manuell jeden einzelnen Schritt durchzuführen, und darauf zu achten alle Projekte beim Zusammenbauen der Software zu berücksichtigen.

GRUNDSTRUKTUR EINES BUILD FILES

```
<project name="uebung06 demo project" default="jar">
    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes" includeantruntime="false"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="at.fhj.itm.demo.HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <echo message="Attempting to run!" />
        <java jar="build/jar/HelloWorld.jar" output="output.txt" fork="true" />
    </target>

</project>
```

Die einzelnen Targets werden ebenso in den Folien dargestellt.

Begriffserklärungen:

project root Element inkl. Attribute, *name* zur Benennung des Build Files und *default* um ein Standard Target anzugeben

target eine einzelne Aufgabe, die selbst definiert werden kann, welche verschiedene Ant Tasks aufrufen kann

task ein Ant Task ist z.B. <jar … /> oder <java … /> wo ein bestimmter Prozess aufgerufen wird, wie z.B. bei <javac … /> das Kompilieren der Java Dateien.

depends einzelne Targets können abhängig sein von anderen Targets, so kann z.B. ein Target angegeben werden, wie bei jar ist compile Voraussetzung. So kann Ant die jeweils abhängigen Targets zuvor aufrufen. Auch mehrere Targets könnten mit depends angegeben werden, so würden diese mit einem Beistrich getrennt in der notwendigen Reihenfolge unter *depends="prepare, compile"* angegeben werden

WICHTIGE ANT TASKS

jar	delete
javac	echo
java	property
mkdir	zip

STEP BY STEP

- Erstellen Sie ein leeres **build.xml** im root Verzeichnis Ihres neuen Projekts.
- Erstellen Sie die folgenden **Targets** mit den beschriebenen Funktionen und Abhängigkeiten.
- **Tipp:** Starten Sie ant <target-name> nach jedem erstellten Target. So finden Sie Fehler schneller und vermeiden langes Fehlersuchen. Achten Sie auf die XML Struktur und mögliche Tippfehler bzw. fehlende geschlossene Tags (häufige Fehlerquellen).

init *erzeugt eine Verzeichnisstruktur für die späteren Aufgaben*

erzeugt die notwendigen Verzeichnisse

clean *löscht die Verzeichnisstruktur und stellt einen Ursprungszustand wieder her*

löscht die mit init erstellten Verzeichnisse

compile *kompiliert Ihre Java Klassen*

Main.java wird kompiliert und samt Package in das Verzeichnis *projekt/build-ant* generiert

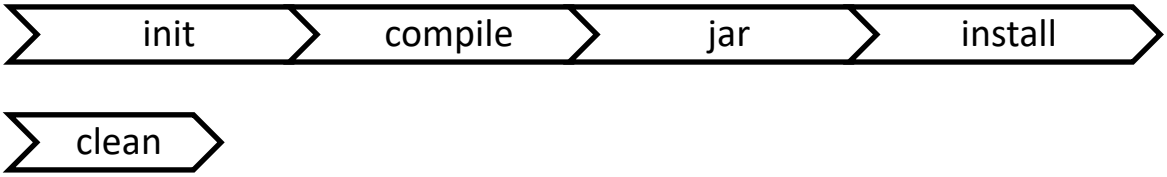
jar *erzeugt eine Jar Datei, die auch ausgeführt werden kann*

erzeugt eine <Dateiname>.jar im Verzeichnis *projekt/lib*

install *kopiert Ihre Jar Datei in ein vordefiniertes Verzeichnis*

kopiert die Jar Datei von *projekt/lib* in das Verzeichnis *projekt/dist*

- Die einzelnen Targets sollen folgende Abhängigkeiten (Dependencies) besitzen.



- Testen Sie Ihr Skript ob die Ausführung funktioniert.
- Versionieren Sie die letzten Änderungen.

LITERATUR

Maven Build

https://github.com/teiniker/teiniker-lectures-configurationmanagement/tree/master/building/Maven-Tutorial

Jar Dateien erzeugen

https://github.com/teiniker/teiniker-lectures-configurationmanagement/tree/master/building/JAR-Tutorial

Einfaches Ant Build.xml

https://ant.apache.org/manual/using.html

Hello World Example inkl. Vergleich Java vs. Ant und Weiterführung des Hello World Examples

http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

Häufig gestellte Fragen bei der Verwendung von ANT

http://ant.apache.org/faq.html