UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Software Engineering im Wintersemester 2021/2022

Prof. Dr. Martin Leucker, Malte Schmitz, Stefan Benox, Julian Schulz, Benedikt Stepanek, Friederike Weilbeer, Tom Wetterich

Übungszettel 2 (Lösungsvorschlag)

06.12.2021

Abgabe bis Donnerstag, 4. November um 23:59 Uhr online im Moodle.

Aufgabe 2.1: Eignung von Vorgehensmodellen

4 Punkte, leicht

Im Folgenden werden verschiedene Projekte beschrieben. Nennen Sie für jedes dieser Projekte ein Vorgehensmodell, das Sie zur Entwicklung des Projektes einsetzen würden und begründen Sie Ihre Auswahl.

1. Es soll ein Rotationsplanungssystem für ein Krankenhaus entwickelt werden, mit dem die Rotationspläne erstellt und verwaltet werden können. Rotationspläne legen fest, wann welcher Mitarbeiter Dienst hat. Für das Projekt sind 20 Personenmonate vorgesehen und die geplante Lebensdauer beträgt 5 Jahre. Der Kunde kann sich noch nicht zwischen verschiedenen Ansichten des Rotationsplans entscheiden und möchte sich erst während der Entwicklung endgültig festlegen.

   ▼ **Lösungsvorschlag**

   Für ein Projekt dieser Art eignen sich generell Prototyp-orientierte Modelle, da hier im Laufe des Projektes zusammen mit dem Kunden konkrete Prototypen evaluiert werden können, um die Anforderungen zu verfeinern.

   Für die Arbeitsorganisation eignet sich zum Beispiel Extreme Programming oder auch Scrum.

Das Spiralmodell ist tendenziell überdimensioniert, da in der kurzen Entwicklungszeit nicht von so vielen vollständigen Iterationen ausgegangen werden kann. Grundsätzlich sind allerdings auch Adaptionen des Spiralmodells für prototypische Vorgehen sehr gut geeignet.

Andere weniger flexible und nicht-iterative Modelle scheiden aus wegen der unklaren Anforderung zum Projektstart.

2. Es soll die Software für ein bemanntes Space Shuttle entwickelt werden. Für das Projekt sind ca. 200 Personenjahre geplant, wobei sich der Umfang im Laufe des Projektes durchaus noch erweitern. Die genauen Anforderungen sollen im Laufe des Projektes durch praktische Erprobungen verfeinert werden. Als Lebensdauer sind 30 Jahre vorgesehen.

   ▼ **Lösungsvorschlag**

   Mit der gleichen Argumentation wie beim vorherigen Beispiel wird auch hier ein prototyp-orientiertes Vorgehensmodell benötigt. Diesmal allerdings in einem deutlich größeren Umfang und in einer tendenziell sicherheitskritischeren Umgebung. Entsprechend eignet sich hier das Spiralmodell sehr gut. Zwischenprodukte während des Prozesses sind sinnvoll um den aktuellen Stand zu evaluieren, das weitere Vorgehen im Detail zu Planen und Scheitern zu verhindern.

   Für die einzelnen Prototyp-Entwicklungen des Spiralmodells lassen sich bei einem Projekt dieser Größe durchaus eigene Vorgehensmodelle anwenden. So kann es insbesondere in früheren Phasen sinnvoll sein, agile Methoden zu verwenden, um möglichst schnell zu ersten Prototypen zu kommen, die zwar nicht produktiv eingesetzt werden können, aber zur Evaluierung der Anforderungen taugen. In späteren Phasen können Konzepte aus dem V-Modell verwendet werden, um durch ausgiebige Tests den Anforderungen der sicherheitskritischeren Umgebung gerecht zu werden.

3. Es soll ein Spiel als App für Smartphones entwickelt werden. Für die Entwicklung sind 10 Personenmonate vorgesehen. Als Lebensdauer sind maximal 3 Jahre vorgesehen. Die Entwickler sind sich noch nicht sicher, wie das Spiel am Ende genau funktionieren und aussehen soll. Es sollen möglichst viele Tests mit potenziellen Kunden vorgenommen werden.

▼ **Lösungsvorschlag**

Für ein Projekt dieser Art bietet sich agile und prototyp-orientierte Modelle an. Im konkreten Fall Extreme Programming: Es stehen wenig Entwickler zur Verfügung und die Anforderungen sind unklar und dynamisch. Es gibt wenig externe Vorgaben und keine besonderen Ansprüche an die Dokumentation oder Sicherheit des Produktes, sodass agile Vorgehensmodelle naheliegen. In der Arbeitsorganisation kann man sich an Scrum orientieren, allerdings ist das Projekt so klein, dass vermutlich der Overhead durch eine vollständige Implementierung des Scrum-Prozesses gar nicht gerechtfertigt ist.

4. Es soll die Software für ein Medizingerät entwickelt werden, das im Operationssaal genutzt wird und den Arzt unterstützt, bei dem Defekte allerdings keine schlimmen Folgen haben, weshalb kein umfassendes Testen und Validieren der Software nötig ist. Die Anforderungen sind bereits vollständig erfasst. Für das Projekt sind 6 Personenjahre vorgesehen und es wird eine Lebensdauer von 15 Jahren für das Produkt angestrebt. Es müssen die Bestimmungen der Medizingerätegesetze eingehalten werden. Das heißt, dass ein extrem strukturierter Entwicklungsprozess stattfinden und eine umfassende Dokumentation erstellt werden muss.

   ▼ **Lösungsvorschlag**

   Durch den vorgeschriebenen strukturierten Entwicklungsprozess sind agile Methoden zwar möglich, müssen aber grundsätzlich immer durch stark strukturierte Dokumentationsprozesse aus dem Lebenszyklusmodell oder dem V-Modell unterstützt werden. Das Spiralmodell ist nicht notwendig, da die Anforderungen klar sind und eine komplexe Validierung in mehreren Phasen nicht vorgesehen ist.

   Als generelles Fazit dieser Aufgabe ergibt sich, dass für konkrete Projekte immer mehrere Vorgehensmodelle geeignet sind. In der Praxis werden fast immer verschiedene Vorgehensmodelle zu einem eigenen Prozess kombiniert. Während es in der theoretischen Untersuchung von Vorgehensmodellen wichtig ist, diese vollständig auszuformulieren, ist es in der praktischen Anwendung üblich, die Konzepte für die eigene Anwendung zu adaptieren und zu mischen. So ist zum Beispiel eine strikte Umsetzung der detaillierten Regeln von Scrum selten sinnvoll, da externe Vorgaben und

Umgebungsbedingungen berücksichtigt werden müssen. Entsprechend ist auch im Scrum-Prozess bereits eine Evaluierung und Optimierung des eigenen Vorgehens vorgesehen, die auch gelebt werden sollte!

Aufgabe 2.2: Anwendungsfalldiagramm

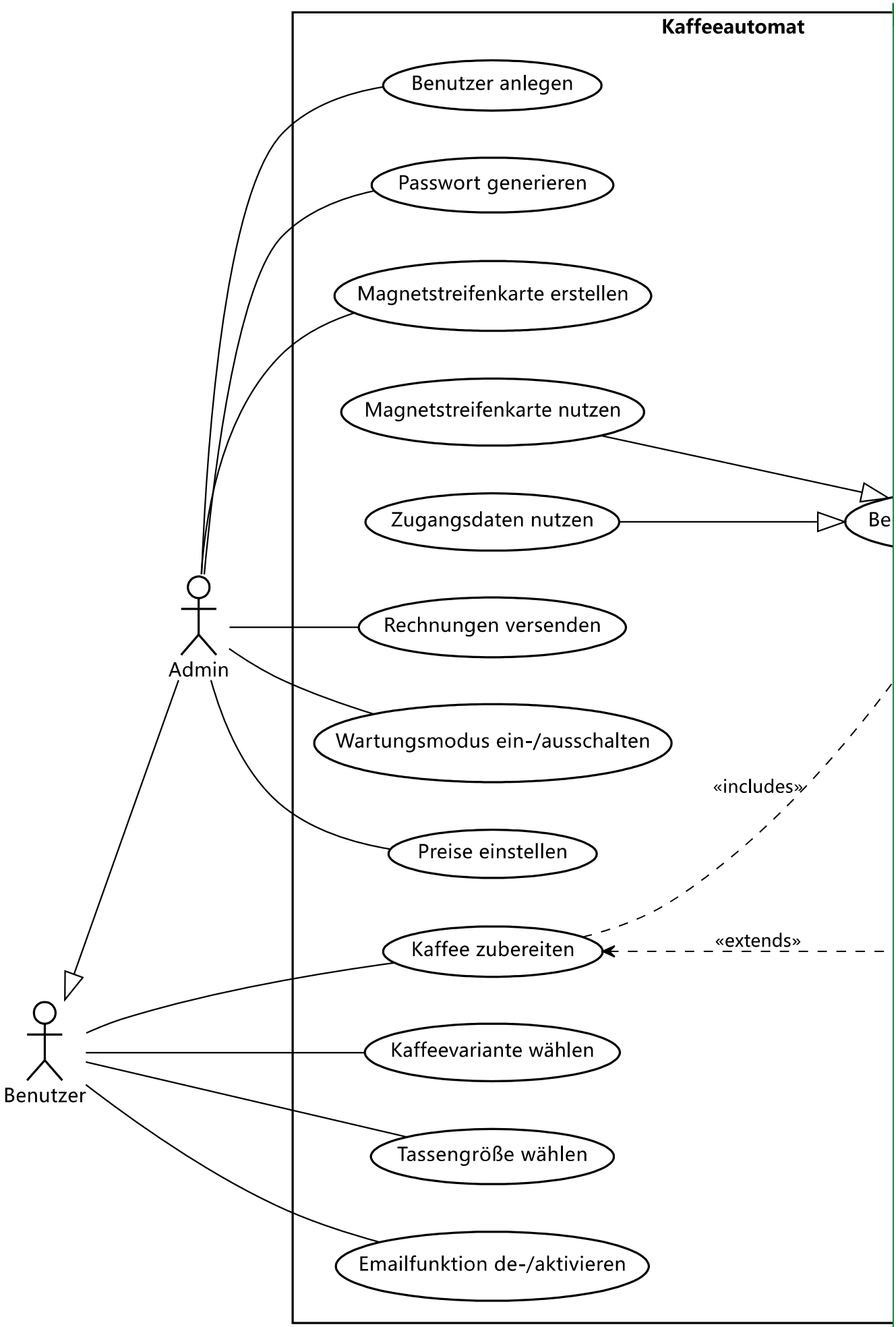
4 Punkte, mittel

In dieser Aufgabe soll ein handelsüblicher Kaffeevollautomat um ein automatisches Abrechnungssystem erweitert werden, das eine bisher geführte Strichliste ersetzt. Das System soll der folgenden Beschreibung entsprechen:

> Der Benutzer kann (wie gewohnt) eine Kaffeevariante und eine Tassengröße auswählen und den gewählten Kaffee zubereiten lassen. Bevor der Kaffee jedoch zubereitet wird, muss sich der Benutzer mithilfe einer Magnetstreifenkarte oder der Eingabe von benutzerspezifischen Zugangsdaten zu erkennen geben. Ein Abrechnungsmodul registriert den Kaffeeverbrauch jedes Benutzers.
>
> Nachdem der Kaffee zubereitet wurde, schickt das System eine E-Mail mit der aktuellen Bilanz an den Benutzer, falls diese E-Mail-Funktion vom Benutzer nicht deaktiviert wurde.
>
> Wenn das System eingeschaltet wird, kann der Verwalter (oder auch Administrator) Benutzer anlegen, Passwörter generieren und Magnetstreifenkarten erstellen. Außerdem kann der Administrator ansonsten die selben Aktionen wie ein herkömmlicher Benutzer ausführen. Der Administrator kann das eingeschaltete System zusätzlich in einen Wartungsmodus versetzen, in dem kein Kaffee mehr zubereitet wird (bezahlter Kaffee wird noch zubereitet, bevor in den Wartungsmodus gewechselt wird). Im Wartungsmodus kann der Administrator z.B. die Preise einstellen.
>
> Am Anfang jeden Monats kann der Administrator auslösen, dass eine Rechnung per E-Mail an alle Benutzer verschickt wird. Benutzer können beim Verwalter beliebige Geldbeträge bar einzahlen, um ihre Bilanz auszugleichen.

Beschreiben Sie die Anwendungsfälle des Gesamtsystems mittels eines UML-Anwendungsfalldiagramms.

▼ Lösungsvorschlag



▶ Quelltext des Diagramms

Aufgabe 2.3: Java-Programmierung

4 Punkte, mittel

Machen Sie diese Aufgabe gründlich und stellen Sie sicher, dass der Code auf Ihrem Computer funktioniert. Wir werden im Laufe des Semesters auf dieses Beispiel zurückkommen.

Schreiben Sie ein Programm, das über die Standard-Eingabe drei Integer-Zahlen entgegen nimmt. Jede Zahl soll dabei als Seitenlänge eines Dreiecks verstanden werden. Das Programm soll anschließend auf der Standardausgabe ausgeben, ob das Dreieck mit diesen drei Seitenlängen gleichschenklig, gleichseitig oder ungleichseitig ist.

Ein Dreieck ist gleichschenklig, wenn es mindestens zwei gleich lange Seiten hat. Ein Dreieck ist gleichseitig, wenn es drei gleich lange Seiten hat. Ein Dreieck ist ungleichseitig, wenn es drei unterschiedlich lange Seiten hat.

Implementieren Sie das beschriebene Programm als Java-Projekt. Nutzen Sie dabei Maven als Build-System und Java-Version 11 als Source- und Target-Version. Ihr Projekt muss sich durch folgende Aufrufe übersetzen und Ausführen lassen:

```
mvn compile
mvn exec:java -Dexec.mainClass="triangle.Main"
```

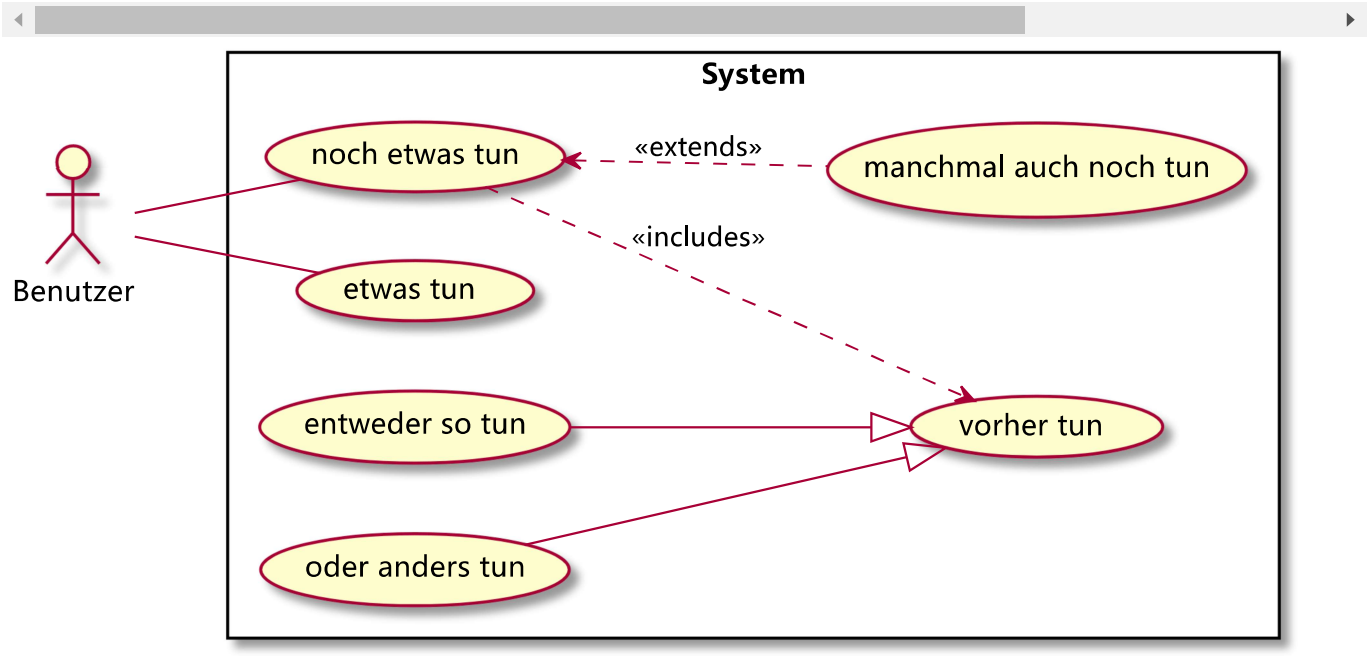
Reichen Sie Ihren Code als Zip-Archiv im Moodle ein. Achten Sie dabei darauf, tatsächlich nur den Quellcode und keine gebauten Artefakte mit abzugeben.

▼ Lösungsvorschlag

Siehe `triangle.zip`.

Hinweise zum Erstellen der Diagramme

UML-Anwendungsfalldiagramme lassen sich sehr elegant mit PlantUML erstellen, was in CodiMD (oder HedgeDoc) direkt eingebunden ist.



▶ Quelltext des Diagramms

Hinweise zur Verwendung von Maven

Maven ist ein Build und Dependency Management Tool, d.h. es kann zum Auflösen von Abhängigkeiten und zum Bauen von Java-Projekten verwendet werden. Maven ist ein Kommandozeilen-Tool, das über eine Datei konfiguriert wird. Diese Datei kann leicht versioniert und ausgetauscht werden, unabhängig von der IDE.

Installation

Im offiziellen Tutorial Maven in 5 Minutes finden sich auch Hinweise zur Installation: Maven ist ein Java-Tool. Es muss also Java installiert sein, damit Maven funktioniert. Zum Beispiel das OpenJDK Temurin in Version 11 oder 17 von Adoptium.

Java ist korrekt installiert, wenn `java -version` die korrekte Version 11 oder 17 im Terminal ausgibt und die Umgebungsvariable `JAVA_HOME` korrekt gesetzt ist, sodass zum Beispiel unter Windows `echo %JAVA_HOME%` folgende Ausgabe im Terminal liefert:

```
C:\Program Files\Java\jdk11.0.11
```

Unter Linux oder Mac kann der Wert der Umgebungsvariable mit `echo $JAVA_HOME` im Terminal geprüft werden. Die Ausgabe kann dann zum Beispiel so aussehen:

```
/usr/lib/jvm/java-11-openjdk-amd64
```

Maven wird als *Binary zip archive* von der offiziellen Download-Seite heruntergeladen. Dieses Archiv wird entpackt, wie in der Installations-Anleitung beschrieben. Das Verzeichnis `bin` muss in die Umgebungsvariable `PATH` hinzugefügt werden.

Ist Maven korrekt installiert, so gibt `mvn -v` in einem neuen Terminal die Version von Maven aus.

Ordnerstruktur

Um Maven zu verwenden sind zwei Dinge notwendig: Die Java-Dateien müssen entsprechend gewisser Regeln in einer Ordnerstruktur abgelegt sein und es braucht eine Konfigurationsdatei `pom.xml`. Wir beginnen mit der Ordnerstruktur:

Angenommen, es gibt mehrere Java-Dateien, also zum Beispiel `Foo.java` und `Bar.java`, die sich beide im Paket `triangle` befinden, dann erwartet Maven folgende Ordnerstruktur:

```
.
├── pom.xml
└── src
    └── main
        └── java
            └── triangle
                ├── Foo.java
                └── Bar.java
```

Es muss sich also im Hauptverzeichnis eine Textdatei `pom.xml` und ein Ordner `src` befinden. Im Ordner `src` befinden sich im Unterordner `main/java` alle Java-Dateien abgelegt entsprechend ihres Pakets.

Konfigurationsdatei

In der `pom.xml` können alle Aspekte des Build-Prozesses und alle Abhängigkeiten deklariert werden. Für die aktuelle Aufgabe brauchen wir keine Abhängigkeiten und es gibt auch keine Besonderheiten im Build-Prozess. Eine minimale `pom.xml` kann also wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<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.
    <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>propositional</groupId>
    <artifactId>propositional</artifactId>
    <version>0.1-SNAPSHOT</version>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Die `groupId` und die `artifactId` müssen gemeinsam einen eindeutigen Bezeichner des Projektes ergeben. Für den Build-Prozess sind sie allerdings unerheblich. Sie werden erst wichtig, wenn Maven-Projekte veröffentlicht werden. Genauso die `version`. Entsprechend müssen diese Werte zwar gesetzt werden, solange Maven allerdings nur als Build-Tool verwendet wird, muss man sich hier nicht all zu viele Gedanken machen.

Die Option `project.build.sourceEncoding` gibt an, welchen Zeichensatz Maven zum Lesen und Schreiben von Dateien verwendet. Alle Dateien sollten immer als UTF-8 gespeichert werden. Das vermeidet Probleme.

Beim `maven-compiler-plugin` wird angegeben, welche Version der Java-Sprache zum Übersetzen verwendet werden soll, also welche Sprach-Features der Compiler versteht, und für welche JVM-Version der Code übersetzt werden soll. Java 11 ist die aktuelle LTS-Version, neuere Versionen sind auch OK.

Verwendung

Wenn die Java-Dateien in den richtigen Ordnern liegen und eine entsprechende `pom.xml` angelegt wurde, dann kann der Quelltext mit folgendem Aufruf kompiliert werden:

```
mvn compile
```

Maven erzeugt dann einen neuen Ordner `target` mit dem Unterordner `classes`, in dem sich die gebauten Class-Dateien befinden.

Über folgenden Befehl können alle von Maven generierten Dateien wieder gelöscht werden. Bitte vor der Abgabe von Quellcode diesen Befehl ausführen, damit die generierten Dateien nicht mit abgegeben werden:

```
mvn clean
```

Um den compilierten Code auszuführen, muss die Main-Klasse explizit angegeben werden. Wenn sich die Main-Klasse direkt im Paket `triangle` befindet, kann sie mit folgendem Befehl ausgeführt werden:

```
mvn exec:java -Dexec.mainClass="triangle.Main"
```

Integrierte Entwicklungsumgebungen (IDEs)

IDEs wie zum Beispiel IntelliJ IDEA können Maven-Projekte importieren. Sie können das Projekt dann auch aus der IDE heraus starten.