Portfolio

Modul 226b

Valentino Rusconi

IAP15b

Inhaltsverzeichnis

[Kompetenzraster 2](#_Toc486433414)

[Spalte B 3](#_Toc486433415)

[1B – Anforderungen und Design 3](#_Toc486433416)

[Anforderung 3](#_Toc486433417)

[Vorwissen 3](#_Toc486433418)

[Reflexion 3](#_Toc486433419)

[2.1B + C – Implementierung 4](#_Toc486433420)

[4B – Methoden-, Sozial-, und Selbstkompetenz 4](#_Toc486433421)

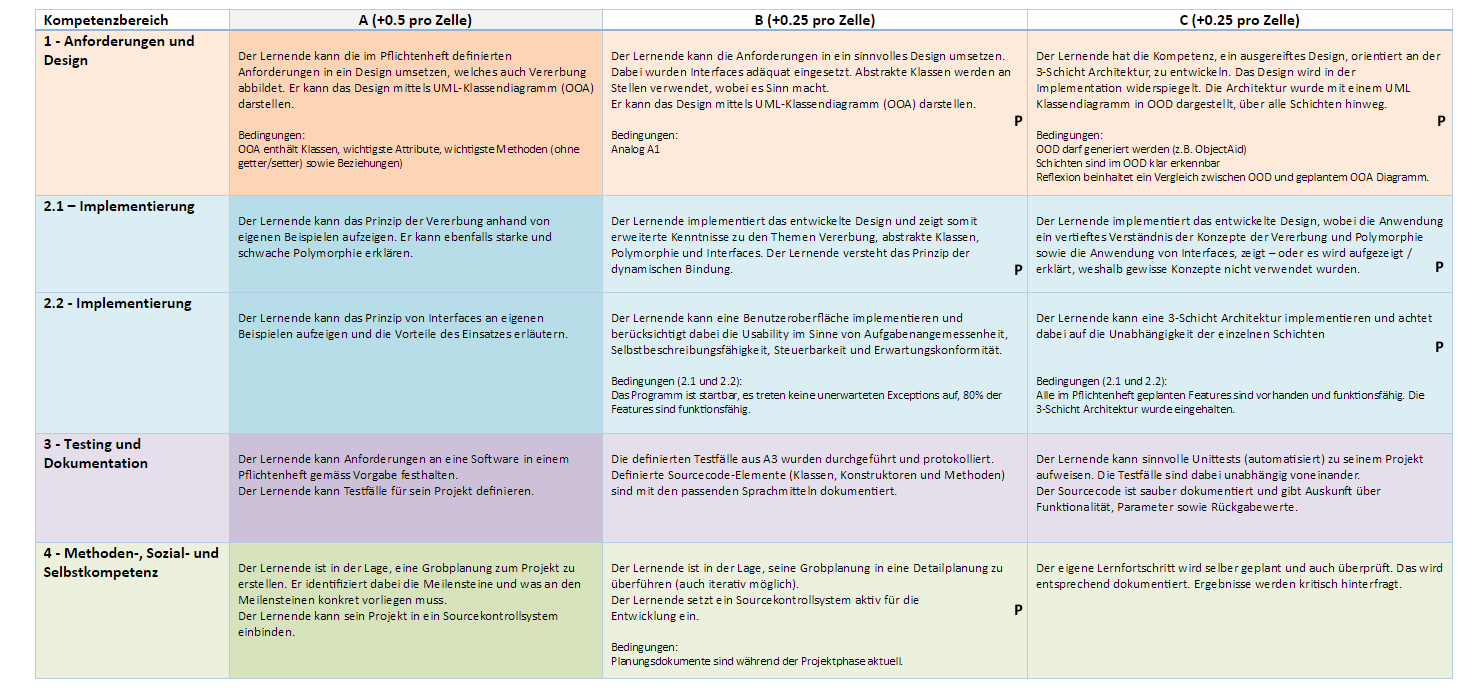
[Spalte C 5](#_Toc486433422)

[1C – Anforderung und Design 5](#_Toc486433423)

[2.2C – Implementierung 6](#_Toc486433424)

[3 Reflexion zum Modul 226 6](#_Toc486433425)

# Kompetenzraster



# Spalte B

## 1B – Anforderungen und Design

### Anforderung

Die gestellte Anforderung war es, ein Klassendiagramm zu erstellen, welches die Basis unserer Projektarbeit darstellt. In einem Klassendiagramm wird grafisch dargestellt, wie die Programmklassen zusammenspielen, und welche Beziehungen bestehen.

### Vorwissen

Durch das Modul 326 und das Modul 226a hatte ich bereits etwas Vorwissen im Bezug auf die Erstellung eines Klassendiagrammes. Daher war mir der Sinn der ganzen Sache bewusst.

### Reflexion

Trotz Vorwissen ist es nochmal etwas ganz Anderes sich von Grund auf das ganze Programm vorzustellen und wie es zusammenspielen soll.   
Anfangs war es etwas schwierig an alles zu denken. Ich wusste nicht mit welcher Schicht ich anfangen soll. Nach und nach ging die Lampe langsam an und es wurde immer verständlicher wie es funktionieren kann.

Beim nächsten Mal sollte ich mir zuerst genaue Gedanken machen wie ich das Programm umsetzten möchte und überlegen wie es am meisten Sinn macht, bevor ich schon mit der Erstellung des UML’s beginne. Damit ich mir diese Zusammenhänge nicht erst während des Erstellens überlegen muss.

## 2.1B + C – Implementierung

**Anforderung**

Die Anforderung bestand darin das entwickelte Design um zu setzen. Somit habe ich nun erweiterte Kenntnisse zu den Themen Vererbung, abstrakte Klassen, Polymorphie und Interfaces.

**Vorwissen**

Ich habe im Geschäft bereits ein ähnliches Tool programmiert. Jedoch habe ich das Design nicht geplant und einfach angefangen zu programmieren. Dadurch wurde mein Code relativ „hässlich“ und ich habe angefangen zu basteln, was mir bei weiteren Anforderungen zum Verhängnis wurde, da ich um die Anforderungen um zu setzten, wieder wo anders anfangen zu basteln musste.

**Reflexion**

Was anfangs etwas schwierig erschien, war die Vorstellung wie genau das nun „korrekt“ programmiert wird, anfangs war die ganze Logik welche sich nun in der Logik Schicht befindet, in der Präsentationsschicht implementiert.

Hier habe ich gelernt Redundanzen zu vermeiden und durch ein geplantes Design die verschiedenen Themen in der Programmierung um zu setzen und besser zu verstehen, wofür sie zu gebrauchen sind. Durch eine abstrakte Klasse kann man Redundanzen verhindern und auch verhindern dass eine Instanz dieser Klasse erzeugt werden kann. Was in unserem Programm von Vorteil ist, da es nicht einfach ein Objekt vom Typ „Artikel“ geben soll, aber von den Tochterklassen. Ausserdem kann durch die Interfaces die Datenschicht einfacher durch eine andere ersetzt werden ohne dass gross etwas angepasst werden muss. Durch die Polymorphie in der Klasse „Article“ kann der Preis individuell in den Unterklassen berechnet werden.

Durch die Planung viel es mir viel einfacher die Implementation umzusetzen.

## 4B – Methoden-, Sozial-, und Selbstkompetenz

**Anforderung**Die Anforderung war es eine Grobplanung für die Projektdauer zu gestalten und diese weiter zu detaillieren in einer Detailplanung.

**Vorwissen**  
Durch die Arbeiten meiner Mitarbeiter hatte ich bereits grobe Erfahrungen gemacht, wie man vorgeht bei einer Grobplanung.

**Reflexion**

Bei der Grobplanung war es relativ einfach, diese zu gestalten, da mir persönlich jedoch noch das Zeitgefühl fehlt, einzuschätzen was wie lange dauert. Dadurch kam es im Zeitplan um einige Verschiebungen im Meilenstein 1 welche im Meilenstein 2 ein neues zu Hause gefunden haben. Jedoch hat mir die Detailplanung dabei geholfen, die Meilensteine nochmals in kleinere «Happen» zu verkleinern um eine noch bessere Übersicht zu haben, wo man gerade steht, und was genau zu tun ist bis zu einem gewissen Zeitpunkt.

# Spalte C

## 1C – Anforderung und Design

**Anforderung**

Die Anforderung besteht darin, dass ein UML Diagramm welches geplant wurde auch so umgesetzt wurde.

**Vorwissen**

Durch das Modul 326 (ÜK) war mir das 3 Schichten Modell schon etwas bekannt, jedoch hatten wir in diesen 5 Tagen nicht die Zeit es uns vertieft anzusehen.

**Reflexion**

Durch die Umsetzung des geplanten Modells habe ich viel gelernt. Vor allem wie man Datenschichten trennt. Bis kurz vor Abgabe ist mir nicht aufgefallen, dass ich die „grosse“ Logik nicht vom View getrennt habe, und somit eine Abhängigkeit bestand. Durch erneutes Hinterfragen, was der grosse Vorteil eines solchen Modelles ist, ging mir die Lampe an und es wurde mir klar, wie es um zu setzten ist. Dadurch dass ich den Sinn dieses Modelles verstanden, war das Resultat nach Fertigstellung des Projektes auch wie geplant.

**Vergleich: geplant – generiert**

Es gab keine grossen Unterschiede, zwischen geplant und generiertem Klassenmodell. Nur im View Package gab es nicht nur eine View, welche auf die Logik Schicht zugreift, sondern mehrere. Die Trennung der Schichten und die Umsetzt konnten wir wie gewollt umsetzten. Die Klassenmodelle sind angefügt im Google Drive Ordner oder im OneNote.

**Anhang**

**-** Diagramme im Google Drive Ordner/OneNote

## 2.2C – Implementierung

**Anforderung**

Die Anforderung bestand darin die 3-Schichten Architektur zu beachten und auf Unabhängigkeiten der einzelnen Schichten zu achten.

**Vorwissen**

Durch das Modul 326, war mir der Sinn bereits etwas bekannt, jedoch war es mir nicht genau klar, was mit erreicht werden will, oder wie genau es umgesetzt wird.

**Reflexion**

Durch das Projekt habe ich sehr viel gelernt, was Architektur betrifft. Ich habe begriffen, warum man Schichten trennt und weshalb man auf Unabhängigkeiten achtet.

Beispielsweise in der Datenschicht möchte man eine Datenbank einsetzten anstelle eines einfachen Arrays, greift man nun über ein Interface darauf zu muss man nur wenig anpassen. Und eine neue Klasse erstellen welche die Datenbank Kommunikation regelt, die Logik Schicht bekommt damit keinen grossen Wind, und ist somit unabhängig von der Datenschicht und kann auch auf andere Art und Weise weiter verwendet werden. Ausserdem muss man nicht überall wieder alles anpassen, wie es wäre wenn man solche Muster nicht beachten würde.

# 3 Reflexion zum Modul 226

Im Modul 226b haben ich viel gelernt. Vor allem im Sinne der Planung eines Projektes. Wie ich bereits erwähnt habe, spielt die Vorbereitung auf die Implementation in meiner Zukunft eine grosse Rolle, weil ich so viele Fehler verhindern kann. Ich werde nicht einfach wieder vor den Computer sitzen und anfangen zu programmieren ohne dass ich mir Gedanken gemacht habe welche Konzepte ich wo und weshalb einsetzten werde. Ausserdem habe ich gelernt wo welche Konzepte der objektorientieren Programmierung einzusetzen sind und weshalb es Sinn macht sie einzusetzen.

Ich habe ausserdem gelernt, das ein Zeitplan eine grosse Rolle spielt, man sollte sich immer im Vorfeld Gedanken machen, was wie viel Zeit beansprucht und es genau planen. Was mir hier sehr geholfen hat, ist die Detailplanung, weil ich so wusste wo ich gerade stehe und was zum erlangen dieses Meilensteines noch fehlt. So bin ich nicht in eine Zone geraten, in der ich nicht wusste, was schon gemacht ist und was bis wann noch gemacht werden muss. Ich habe also den Faden nie verloren, das werde ich mir in Zukunft auch versuchen anzugewöhnen.

Wie ich auch bereits schon erwähnt habe werde ich auch darauf, dass die Struktur eines Programmes, zumindest im Groben, schon vor der Implementation überdacht wurde. In Vergangenheit habe ich dies nicht beachtet und habe angefangen zu basteln, was den Code unschön macht und die Erweiterbarkeit im schlimmsten Falle beinträchtig.