DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

„Mensch ärgere Dich nicht“

Programmentwurf

in der Vorlesung „Advanced Software Engineering“

im fünften und sechsten Semester

des Studienganges Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Nadine Weiß und Manuel Berg

31. 05. 2022

Matrikelnummern	5931590
Kurs	TINF19B5
Dozent	Maurice Müller

Eidesstattliche Erklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 09. 2017)

Ich versichere hiermit, dass ich mein Programmentwurf vom 31. 05. 2022 mit dem Thema „Mensch ärgere Dich nicht“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum Unterschrift

Hinweis: Der LaTeX-Quellcode dieses Dokumentes basiert auf der „Vorlage für Berichte der DHBW Karlsruhe“, welche freundlicherweise von Prof. Dr. Jürgen Vollmer zur Verfügung gestellt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
GI	Gesellschaft für Informatik e.V.

1. Einführung (4P)

1.1 Übersicht über die Applikation (1P)

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

1.2 Wie startet man die Applikation? (1P)

[Wie startet man die Applikation? Was für Voraussetzungen werden benötigt? Schritt-für-Schritt- Anleitung]

1.3 Technischer Überblick (2P)

[Nennung und Erläuterung der Technologien (z.B. Java, MySQL, ...), jeweils Begründung für den Einsatz der Technologien]

2. Clean Architecture (8P)

2.1 Was ist Clean Architecture? (1P)

[Allgemeine Beschreibung der Clean Architecture in eigenen Worten]

2.2 Analyse der Dependency Rule (2P)

[1 Klasse, die die Dependency Rule einhält und 1 Klasse, die die Dependency Rule verletzt; jeweils UML der Klasse und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

Positiv-Beispiel: Dependency Rule

Negativ-Beispiel: Dependency Rule

2.3 Analyse der Schichten (5P)

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML der Klasse (ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

Schicht: [Name]

Schicht: [Name]

3. SOLID (8P)

3.1 Analyse SRP (3P)

[Jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML der Klasse und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

Positiv-Beispiel

Negativ-Beispiel

3.2 Analyse OCP (3P)

[Jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML der Klasse und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

Positiv-Beispiel

Negativ-Beispiel

3.3 Analyse LSP/ISP/DIP (2P)

[Jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP); jeweils UML der Klasse und Begründung, warum man hier das Prinzip erfüllt/nicht erfüllt wird]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

Positiv-Beispiel

Negativ-Beispiel

4. Weitere Prinzipien (8P)

4.1 Analyse GRASP: Geringe Kopplung (4P)

[jeweils eine bis jetzt noch nicht behandelte Klasse als positives und negatives Beispiel geringer Kopplung; jeweils UML Diagramm mit zusammenspielenden Klassen, Aufgabenbeschreibung der Klasse und Begründung warum hier eine geringe Kopplung vorliegt bzw. Beschreibung, wie die Kopplung aufgelöst werden kann]

Positiv-Beispiel

Negativ-Beispiel

4.2 Analyse GRASP: Hohe Kohäsion (2P)

[eine Klasse als positives Beispiel hoher Kohäsion; UML Diagramm und Begründung, warum die Kohäsion hoch ist]

4.3 DRY (2P)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher); begründen und Auswirkung beschreiben]

5. Unit Tests (8P)

5.1 10 Unit Tests (2P)

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test	Beschreibung

Tabelle 5.1: Übersicht der 10 Unit Tests

5.2 ATRIP: Automatic (1P)

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]

5.3 ATRIP: Thorough (1P)

[Code Coverage im Projekt analysieren und begründen]

5.4 ATRIP: Professional (1P)

[jeweils 1 positves und negatives Beispiele zu 'Professional'; jeweils Code-Beispiel, Analyse und Begründung, was professionell/nicht professionell an den Beispielen ist]

5.5 Fakes und Mocks (1P)

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten; zusätzlich jeweils UML Diagramm der Klasse]

6. Domain Driven Design (8P)

6.1 Ubiquitous Language (2P)

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichnung	**Bedeutung**	**Begründung**

Tabelle 6.1: 4 Beispiele für die Ubiquitous Language

6.2 Repositories (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.3 Aggregates (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.4 Entities (1,5P)

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

6.5 Value Objects (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist]

7. Refactoring (8P)

7.1 Code Smells (2P)

[jeweils 1 Code-Beispiel zu 2 unterschiedlichen Code Smells aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

7.2 2 Refactorings (6P)

[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

8. Entwurfsmuster (8P)

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

Entwurfsmuster: [Name] (4P)

Entwurfsmuster: [Name] (4P)

Literaturverzeichnis