DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

RaHM-Lab
Positionsregelung Drohne

Projekt Software Engineering 2

für die Prüfung zum
Bachelor of Science

im Studiengang Informatik
an der DHBW Karlsruhe

von
Michael Maag
Theresa Uhlmann
17.05.2022

Gutachter der DHBW Karlsruhe Daniel Lindner

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

USB	Universal Serial Bus

Tabellenverzeichnis

Code-Verzeichnis

1 Einleitung

“PLACE CITE HERE “

[?]

Diese Ausarbeitung wird als Prüfungsleistung für das Modul *Software Engineering 2* angefertigt. Die Inhalte basieren maßgeblich auf der zugrundeliegenden Vorlesung. Die Prüfungsleistung wird auf entstandenem Code einer Studienarbeit ausgeführt.

Anmerkung: Für diese Ausarbeitung werden fachliche Begrifflichkeiten vorausgesetzt, sofern diese nicht innerhalb der Ausarbeitung erklärt werden. Sind Begriffe für Lesende unklar, sind diese an geeigneter Stelle nachzuschlagen. Auf eine voranstehende Erklärung aller genutzten und nicht näher erklärten Begriffe wird in dieser Ausarbeitung verzichtet, um den Rahmen dieser Arbeit einhalten zu können.

2 Problemstellung

Diese Ausarbeitung baut auf dem Code einer Studienarbeit auf.

Die Problemstellung der zugrundeliegenden Studienarbeit lautet wie folgt: „

BESCHREIBUNG! *Text hier einfügen!*

“[]

3 Methodik

4 Domain Driven Design

BESCHREIBUNG! *Sem5 VL7*

4.1 Ubiquitous Language

BESCHREIBUNG! *Sem5 VL7-1 Min64*

4.2 Repositories

BESCHREIBUNG! *Sem5 VL9-1 Min12*

4.3 Aggregates

BESCHREIBUNG! *Sem5 VL8-2 Min33*

BESCHREIBUNG! *Sem5 VL9-1 Min6*

4.4 Entities

BESCHREIBUNG! *Sem5 VL7-3 Min46*

BESCHREIBUNG! *Sem5 VL8-1 Min84*

4.4.1 Surrogatschlüssel

BESCHREIBUNG! *Sem5 VL8-2 Min7*

Mögliches Beispiel: Sequenz in ROS-Messages.

Weiterhin wird der Zeitstempel mitgeführt. Allferdings nicht mit dem Zweck als Schlüssel.

4.5 Value Objects

BESCHREIBUNG! *Sem5 VL7-3 Min35*

BESCHREIBUNG! *Sem5 VL7-3 Min49*

BESCHREIBUNG! *Sem5 VL8-1 Min36*

5 Programming Principles

5.1 SOLID

BESCHREIBUNG! *Sem5 VL4*

5.1.1 Single Responsibility Principle

5.1.2 Open/Closed Principle

5.1.3 Liskov Substitution Principle

Hier als Beispiel das Controller-Array in ControllerSystem

5.1.4 Interface Segregation Principle

5.1.5 Dependency Inversion Principle

5.2 GRASP

5.2.1 Kopplung

5.2.2 Kohäsion

5.3 DRY

Don't repeat yourself

Aus aufwändiger Domain-Architektur ergibt sich, dass das umgesetzt wurde => viele kleine Klassen.

5.4 KISS

BESCHREIBUNG! *Sem5 VL6*

5.5 YAGNI

BESCHREIBUNG! *Sem5 VL6-2 Min36*

You ain't gonna need it

Erst Code hinzufügen, wenn benötigt.

Wurde im Projekt teilweise durchgeführt - git-Historie durchschauen... Fuck ist das aufwändig... -.-

Mein allgemeiner Programmier-Stil entspricht nicht vollständig diesem Prinzip. Grundlegende Funktionalitäten werden hinzugefügt, WEIL sie benötigt werden, nicht WENN es soweit ist :D Immerhin weiß man ja schon so grob, was alles im Projekt benötigt wird... Zugegeben, hier wird oft erst die leere Funktion eingebaut und der benötigte Code dann, wenn es an den jeweiligen Test/Ausprobieren geht. Beispiel: Vector3D::rotate() Body

5.6 Conway's Lay

BESCHREIBUNG! *Sem5 VL6-2 Min65*

6 Architektur

7 Entwurfsmuster

8 Software Tests

Es gibt folgende Test-Arten:

- Unit Test
- Integration Test
- System Test
- Acceptance Test
- BESCHREIBUNG! *fehlt hier was?*

Allerdings wird in dieser Ausarbeitung nur auf Unit Tests eingegangen.

8.1 Unit Tests

Wahl fällt auf *banditcpp*, da es sich hierbei um ein *headers only* Framework handelt. Daraus folgt, dass die Test-Umgebung nicht in den Produktiv-Code eingebunden werden muss. Es kann quasi als PlugIn „aufgesetzt“ werden.

BESCHREIBUNG! *Bandit bietet leider keine Code Coverage...*

8.1.1 Umgesetzte Unit Tests

8.1.2 Code Coverage

9 Refactoring

10 Fazit und Ausblick

Literaturverzeichnis

[1] Intel's First Microprocessor: Its invention, introduction, and lasting influence,
online, https://www.intel.de/content/www/de/de/history/museum-story-of-intel-4004.html
veröffentlicht -unbekannt-, verändert 08.04.2020, abgefragt 06.09.2021

[2] Saks D., Better even at the lowest levels,
online, https://www.embedded.com/better-even-at-the-lowest-levels/
veröffentlicht 01.11.2008, verändert 05.12.2020, abgefragt 28.07.2021

[3] Application Note Object-Oriented Programming in C,
online, https://www.state-machine.com/doc/AN_OOP_in_C.pdf
veröffentlicht 06.11.2020, abgefragt 28.07.2021

[4] Kirk N., How do strings allocate memory in c++?,
online, https://stackoverflow.com/questions/18312658/how-do-strings-allocate-memory-in-c
veröffentlicht 19.08.2013, abgefragt 17.08.2021

[5] Bansal A., Containers in C++ STL (Standard Template Library),
online, https://www.geeksforgeeks.org/containers-cpp-stl/
veröffentlicht 05.03.2018, verändert 12.07.2020, abgefragt 17.08.2021

[6] Automatic Storage Duration,
online, https://www.oreilly.com/library/view/c-primer-plus/9780132781145/ch09lev2sec2.html
veröffentlicht -unbekannt-, abgefragt 17.08.2021

[7] Noar J., Orda A., Petruschka Y., Dynamic storage allocation with known durations,
online, https://www.sciencedirect.com/science/article/pii/S0166218X99001754
veröffentlicht 30.03.2000, abgefragt 17.08.2021

Anmerkung: Wird hier ein Veröffentlichungsdatum als "-unbekannt-" markiert, so konnte diese Angabe weder auf der entsprechenden Webseite, noch in deren Quelltext ausfindig gemacht werden.

Anhang