

Entwicklung einer Hardware-Health-Monitoring Lösung für Pepperl+Fuchs
HMI Systeme

Bachelorarbeit

des Studienganges Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Vitali Mostowoj

018.09.2023

Bearbeitungszeitraum:	12 Wochen
Matrikelnummer, Kurs:	9960312, Tel20 At1
Ausbildungsfirma, Abteilung:	Pepperl + Fuchs SE, HMI
Standort:	Lilienthalstraße 200, 68307 Mannheim
Betreuer der Ausbildungsfirma:	Dr. Marc Seissler
Gutachter der Dualen Hochschule:	Prof. Dr. Joachim Priesnitz

Unterschrift (Betreuer)

Todo list

mehr auf HMI eingehen, warum braucht man diese, wo werden die eingesetzt 1

Sperrvermerk

„Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung des Dualen Partners vorliegt.“ [Ende der Sperrfrist: 31.12.2222]

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: Entwicklung einer Hardware-Health-Monitoring Lösung für Pepperl+Fuchs HMI Systeme selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Abstract

Inhaltsverzeichnis

Inhaltsverzeichnis	III
Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
1 Einleitung	1
1.1 Pepperl + Fuchs / HMI	1
1.2 Problemstellung und Ziel der Arbeit	2
1.3 Anforderungen	2
2 Stand der Technik	3
2.1 Vergleich von Computer Monitoring Software auf dem Markt	3
2.1.1 Computer Monitoring Software	3
2.1.2 HwiNFO	3
2.2 Produktfamilie VisuNet	3
2.2.1 VisuNet FLX & GXP	3
2.2.2 VisuNet RM Shell & Control Center	3
2.3 Software Design Konzepte	4
2.3.1 Adapter Pattern	4
2.3.2 Strategie Design Pattern	5
2.4 Datenbanken	7
2.4.1 SQL - Structured Query Language	8
2.4.2 SQLite Embedded Datenbank	8
2.5 MTBF und Reliability	10
3 Architektur Konzept	11
3.1 Datenerfassung	12
3.1.1 Entwurf einer Architektur zum Auslesen der Systemhardware	12

3.1.2	Entwurf eines Datenbankmodells zum Speichern der Messwerte . . .	12
3.2	Datenverarbeitung	12
3.2.1	Health Status Definition	12
3.2.2	Ermittlung des Health Status	12
3.2.3	Ermittlung der Systemstatus Historie	12
3.2.4	Ermittlung der Systemzuverlässigkeit	12
3.3	Dashboard	12
3.3.1	Bereitstellung der Daten	12
3.3.2	Konzeptentwurf zur Datenvisualisierung	12
3.4	Entwurf eines Task Scheduling Verfahrens	12
4	Prototypische Implementierung	13
4.1	Implementierung des Taskschedulers	14
4.2	Implementierung einer Plattform unabhängigen Datenerfassung	14
4.2.1	Umsetzung der Hardware Services	14
4.2.2	Umsetzung der Datenbank Services	14
4.2.3	Umsetzung der Strategien zur Datenerfassung	14
4.3	Implementierung der Datenverarbeitung	14
4.3.1	Implementierung der Algorithmen zur Health Status Erfassung . . .	14
4.4	Implementierung der Datenvisualisierung	14
4.4.1	Implementierung der API	14
4.4.2	Aufbau eines Dashboards	14
5	Ausblick	15
	Literaturverzeichnis	I

Abbildungsverzeichnis

1.1	Standorte der Pepperl+Fuchs SE	1
2.1	Adapter Pattern Struktur [1]	5
2.2	Strategie Pattern Struktur [1]	6
2.3	Volumen der weltweit generierten Daten bis 2027 [3]	7
2.4	SQL Befehls Kategorien [4]	9

Tabellenverzeichnis

Abkürzungsverzeichnis

P+F Pepperl+Fuchs

HMI Human-Machine-Interface

SQL Structured Query Language

SEQUEL Structured English Query Language

1 Einleitung

1.1 Pepperl + Fuchs / HMI

Die Pepperl+Fuchs (P+F) wurde 1945 von Walter Pepperl und Ludwig Fuchs gegründet. Anfangs war sie eine Radioreperaturwerkstatt, welche sich erst nach der Entwicklung eines eigenen Näherungsschalters so wie eines eigensicheren Transistorverstärkers auf das Gebiet der Elektronik ausweitete. Inzwischen entwickelt, produziert und vertreibt P+F Baugruppen und Sensoren für den Automatisierungsmarkt.



Abbildung 1.1: Standorte der Pepperl+Fuchs SE

Pepperl Im Bereich der Prozessautomation ist P+F führender Hersteller industrieller

mehr auf
HMI ein-
gehen,
warum
braucht
man die-
se, wo

Sicherheitsausstattungen. Das Produktportfolio umfasst Trennbarrieren, Signaltrenner, Zener-Barrieren, Feldbus-Technologien, Remote-I/O, HART-Interface-Lösungen, Mensch-Maschine-Schnittstellen (HMI) für Gefahrenbereiche, Füllstandsüberwachung, Überdruckkapselungssysteme, Schaltschränke, Feldverteiler und Warnsysteme für Ex-Umgebungen. Die Abteilung *Human Machine Interfaces* beschäftigt sich dabei mit der Entwicklung von Industrieller Computer Hard- und Software. Dies reicht von zubehör bishin zu vollständige Systemen mit Ex-Umgebungen Zertifizierung. Ziel der Systeme ist die Überwachung und Steuerung sämtlicher Produktionsschritte.

1.2 Problemstellung und Ziel der Arbeit

1.3 Anforderungen

2 Stand der Technik

2.1 Vergleich von Computer Monitoring Software auf dem Markt

2.1.1 Computer Monitoring Software

2.1.2 HWiNFO

2.2 Produktfamilie VisuNet

Für

2.2.1 VisuNet FLX & GXP

2.2.2 VisuNet RM Shell & Control Center

2.3 Software Design Konzepte

Eine solide Softwarerchitektur ist entscheidend für die erfolgreiche Entwicklung und Wartung eines Programmes. Sie legt den Grundstein für die anschließende Implementierung. Durch eine gute Architektur wird sichergestellt, dass das Programm skalierbar, effizient, robust und gut zu warten ist.

Hierbei bieten sogenannte Design Patterns Abhilfe. *Jedes Muster beschreibt zunächst ein in unserer Umwelt immer wieder auftretendes Problem, beschreibt dann den Kern der Lösung dieses Problems, und zwar so, dass man diese Lösung millionenfach anwenden kann, ohne sich je zu wiederholen* (Christof Alexander *Eine Muster-Sprache* [Löcker Verlag, Wien, 1995, Seite x]). Diese Definition für Muster bezieht sich auch auf objektorientierte Design Patterns. Das Verwenden dieser Patterns ermöglicht Entwicklern, von der Erfahrung anderer zu profitieren, um bereits gelöste Probleme nicht nochmal lösen zu müssen. Zudem steigern sie auch die Codequalität. Der Code wird lesbarer und die Wartung desselben wird leichter. Zudem wird auch die Implementierung neuer Erweiterungen und das Einbringen in die Software durch gängige Design Patterns erleichtert. [1, S.25 ff]

Alle gut strukturierten objektorientierten Architekturen basieren auf Mustern (Grady Booch [1, S.21]). In den folgenden Kapiteln wird genauer auf die in dieser Arbeit verwendeten Design Patterns eingegangen.

2.3.1 Adapter Pattern

Zweck des Adapter Patterns ist die Anpassung der Schnittstelle einer Klasse an eine andere von dem Client erwartete Schnittstelle. Somit ermöglicht das Pattern die Zusammenarbeit von zwei Klassen, welche aufgrund ihrer Schnittstellen nicht möglich wäre. Das Adapter Pattern ist auch unter dem Namen Wrapper bekannt, welcher im folgenden Verlauf der Arbeit verwendet wird.

Das Pattern kommt immer dann zum Einsatz, wenn eine bereits existierende Klasse genutzt werden soll, jedoch die Schnittstelle der Klasse nicht mit den aktuellen Anforderungen des Clients übereinstimmt. Des Weiteren wird das Pattern verwendet, wenn eine wiederverwendbare Klasse erzeugt werden soll, welche mit unabhängigen und nicht vorhersehbaren Klassen interagieren soll.

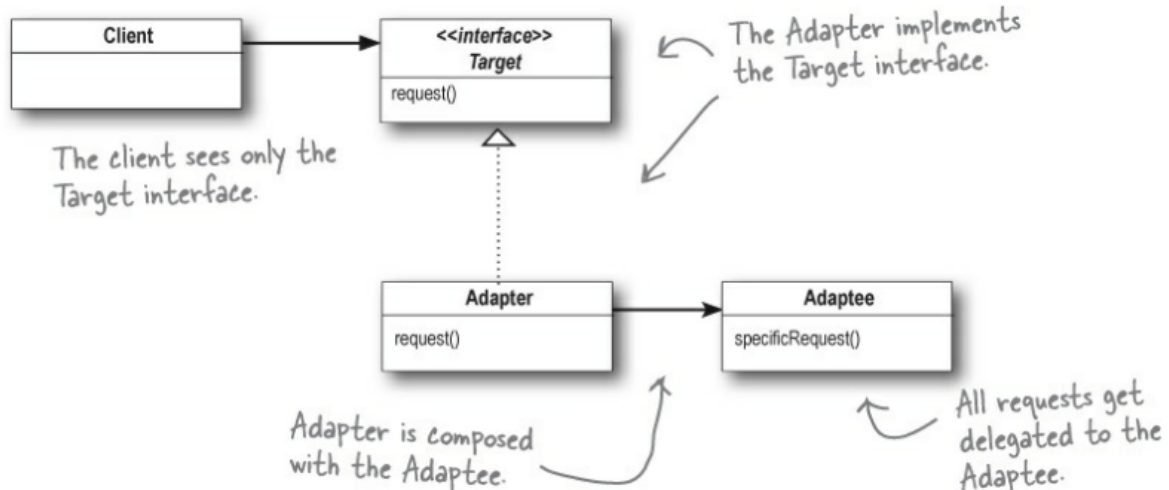


Abbildung 2.1: Adapter Pattern Struktur [1]

Das Design Pattern besteht aus einem *Target*, welches die vom Client verwendete Schnittstelle definiert. Zu dem kommt der *Client*, welcher mit den Objekten zusammen arbeitet, die der Zielschnittstelle entsprechen. Zuletzt beinhaltet das Adapter Pattern einen *Adaptee* so wie den *Adapter* selbst. Der *Adaptee* definiert eine bestehende Schnittstelle, welche vom *Adapter* adaptiert werden muss.

Der *Client* ruft die gewünschte Operation auf einer *Adapter*-Instanz auf, welche anschließend die gewünschten *Adaptee*-Operation ausführt.

2.3.2 Strategie Design Pattern

Zweck des Strategy (Strategie) Patterns ist es, eine Familie von einzelnen gekapselten und Austauschbaren Algorithmen zu schaffen. Dieses Pattern ermöglicht eine variable und vom Client unabhängige Nutzung des Algorithmus.

Das Pattern kommt zum Einsatz wenn eine Reihe von zusammenhängenden Klassen sich nur in Ihrem Verhalten unterscheiden, verschiedene Varianten eines Algorithmus erfordert werden, der Client keine Kenntnis von den vom Algorithmus verwendeten Daten haben soll, oder eine Klasse verschiedene Verhaltensweisen aufweist.

Das Design Pattern besteht aus den folgenden Teilnehmern. Die *Strategy*, welche eine gemeinsame Schnittstelle für die verwendeten Algorithmen deklariert. Einer oder mehreren *ConcreteStrategy*, welche die Implementierung der Algorithmen oder Klassen ist, so wie

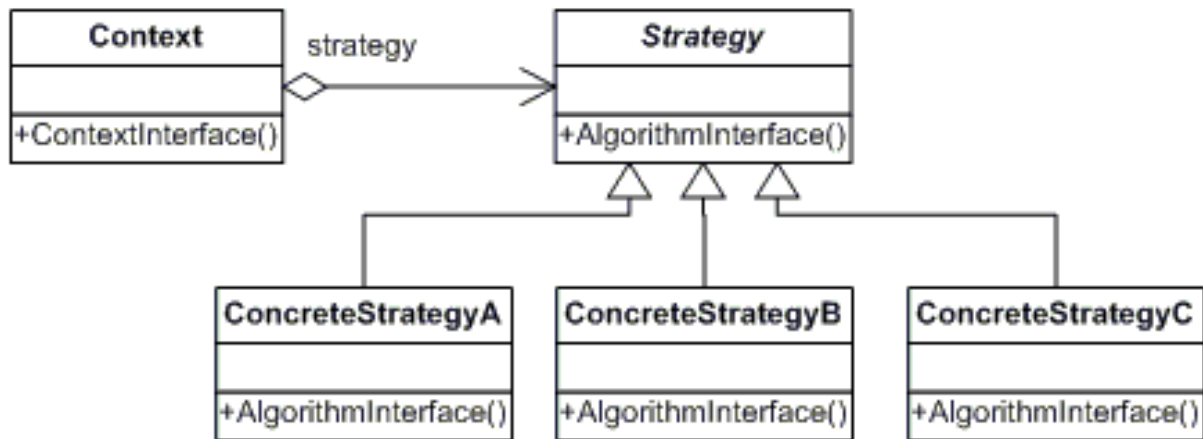


Abbildung 2.2: Strategie Pattern Struktur [1]

dem *Context*, welcher mit einer *ConcreteStrategy* ausgestattet wird. Desweiteren besitzt der *Context* eine Referenz auf das *Strategy* Objekt.[1, S.383 ff]

Über den *Context* kann anschließend zur Laufzeit des Programmes die benötigten *ConcreteStrategy* geladen und ausgeführt werden. Ein konkretes Beispiel hierzu wird im Buch [2, Head First Design Patterns] behandelt, was den Nutzen dieses Patterns nochmal verdeutlicht.

2.4 Datenbanken

Weltweit wurden im Jahr 2022 Daten im Umfang von 103.66 Zettabyte erfasst. Diese Zahl wird sich laut Statistik 2.3 bis zum Jahr 2026 verdoppelt haben. Angesicht dieser Zahlen, sind Datenbanken aus der heutigen Zeit nicht weg zu denken. Sie bieten eine Möglichkeit, große Mengen an Daten Strukturiert abzuspeichern und anschließend auszuwerten.

Hierbei werden Datenbanken Grundsätzlich in Zwei Kategorien unterteilt. Relationale Datenbank und "Nicht relationale Datenbanken". Unterschiede der Datenbankarten machen sich in der Sprache zum Auswerten der DB, ihrer Skalierbarkeit, der Struktur, der Eigenschaften und der Unterstützung durch die Community bemerkbar.

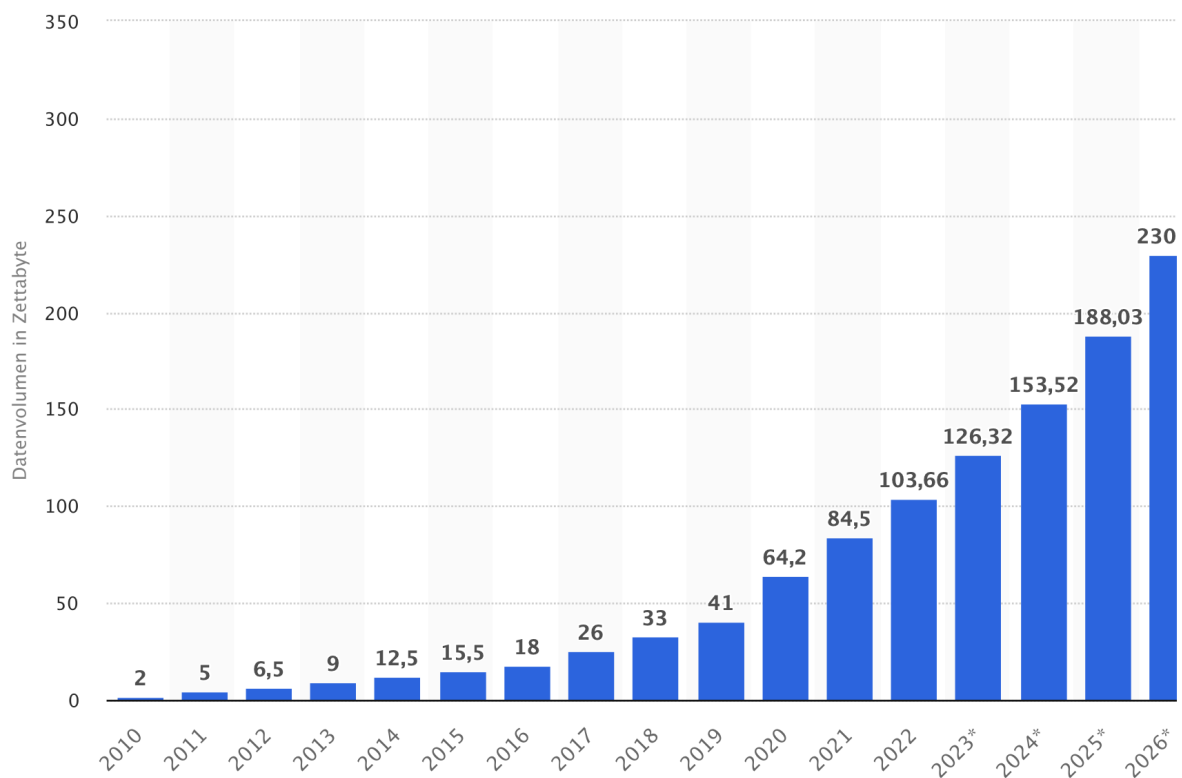


Abbildung 2.3: Volumen der weltweit generierten Daten bis 2027 [3]

2.4.1 SQL - Structured Query Language

IBM-Forscher Edgar F. Codd definierte 1969 ein Datenbankmodell für Relationale Datenbanken. Auf grundlagen seiner Forschung began, in den folgenden Jahren, die entwicklung der Sprache Structured English Query Language (SEQUEL). Codd's Modell für basiert auf der zuordnung von Schlüsseln. Nach einigen Überarbeitungen der implementierung wurde diese anschließend in Structured Query Language (SQL) umbenannt.

SQL ermöglicht insbesondere die Speicherung, Bearbeitung so wie eine Abfrage von Daten in einer Datenbank. Mithilfe des Prinzips der Schlüssel, können Datensätze miteinander verknüpft werden. Somit kann einem Benutzernamen beispielsweise ein echter Name, eine Telefonnummer und eine Email-Adresse zugewiesen werden.

Die besondere eigenschaft von SQL ist das Konzepte von Arrays. Relationale Datenbanken bestehen aus Arrays, welche sich mit Hilfe von verschiedenen Befehlen erzeugen und bearbeiten. **sql**

SQL bietet eine reihe von Befehlen, welche die Interaktion mit der Datenbank ermöglichen. Diese können Grundsätzlich in 5 Kategorien eingeteilt werden (siehe Abb. 2.4). Die wichtigsten Befehle sind dabei *INSERT*, *UPDATE* und *DELETE*, mit welchen sich datensätze schreiben und bearbeiten lassen. Zudem der kommt der *SELECT* Befehl, welcher das auslesen von Datensätzen ermöglicht. Um die tabellenstruktur der Datenbank zu bearbeiten kommen die Befehle *CREATE* und *DROP* zum einsatz. [4]

Natürlich bietet die Programmiersprache eine weit aus komplexere Syntax, um datensätze sortiert auswerten zu können. Eine vollständige dokumentation der Sprache findet sich auf der w3school webseite [5].

2.4.2 SQLite Embedded Datenbank

In der Vorarbeit zu dieser Bachelorarbeit wurde bereits eine auswahl für eine Datenbank getroffen. Dabei wurde sich nach einigen vergleichen für die SQLite Embedded Datenbank Engine entschieden.

Diese Bietet eine zuverlässige, kleine, schnelle und vollfunktionale Datenbank Engine, welche vollständige in das Gesamtsystem integriert werden kann [6]. Zur implementierung der Datenbank in die anwendung wird edie System.Data.SQLite bibliothek für C# verwendet.

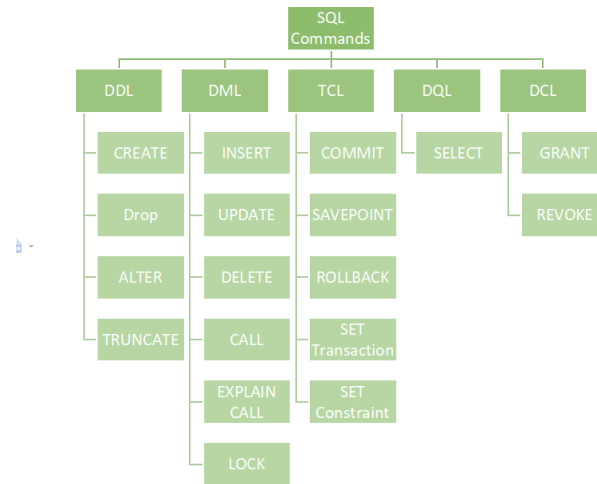


Abbildung 2.4: SQL Befehls Kategorien [4]

2.5 MTBF und Reliability

Es gibt viele Ursachen, welche zu einem Ausfall elektronischer Komponentetn in einem system, führen können. Laut dem Technischen Bericht [7] ist in 50% der Fälle die Temperatur der Komponenten für einen Ausfall verantwortlich. Dies liegt an den unterschiedlichen thermischen Ausdehnungskoeffizienten des Materials auf der Platine haben. Durch die unterschiedliche Ausdehnung der Bauteile und der Platine selbst, kommt es zu zu hohen belastungen der Lötstellen. Während sich dieser Zyklus wiederholt können Risse in den Verbindungen entstehen und ausbreiten. Diese können anschließend zu einem Bruch im elektrischen Stromkreis führen.

3 Architektur Konzept

3.1 Datenerfassung

3.1.1 Entwurf einer Architektur zum Auslesen der Systemhardware

3.1.2 Entwurf eines Datenbankmodells zum Speichern der Messwerte

3.2 Datenverarbeitung

3.2.1 Health Status Definition

3.2.2 Ermittlung des Health Status

3.2.3 Ermittlung der Systemstatus Historie

3.2.4 Ermittlung der Systemzuverlässigkeit

3.3 Dashboard

3.3.1 Bereitstellung der Daten

3.3.2 Konzeptentwurf zur Datenvisualisierung

3.4 Entwurf eines Task Scheduling Verfahrens

4 Prototypische Implementierung

4.1 Implementierung des Taskschedulers

4.2 Implementierung einer Plattform unabhängigen Datenerfassung

4.2.1 Umsetzung der Hardware Services

4.2.2 Umsetzung der Datenbank Services

4.2.3 Umsetzung der Strategien zur Datenerfassung

4.3 Implementierung der Datenverarbeitung

4.3.1 Implementierung der Algorithmen zur Health Status Erfassung

4.4 Implementierung der Datenvisualisierung

4.4.1 Implementierung der API

4.4.2 Aufbau eines Dashboards

5 Ausblick

Literaturverzeichnis

- [1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns - Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp Verlags GmbH & Co.KG, 2015, ISBN: 978-3-8266-9700-5.
- [2] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, *Head First Design Patterns - A Brain Friendly Guide*. O'Reilly Media, Incorporated, 2004, ISBN: 978-0-5960-0712-6.
- [3] „Volumen der jährlich generierten/replizierten digitalen Datenmenge weltweit von 2010 bis 2022 und Prognose bis 2027“. (2023), Adresse: <https://datascientest.com/de/sql-alles-uber-die-datenbanksprache#:~:text=SQL%20oder%20E2%80%9Estructured%20Query%20Language,darin%20enthaltenen%20Daten%20zu%20verwalten.%7D> (besucht am 11.08.2023).
- [4] „SQL | DDL, DQL, DML, DCL and TCL Commands“. (2023), Adresse: <https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/%7D> (besucht am 11.08.2023).
- [5] „SQL Tutorial“. (2023), Adresse: <https://www.w3schools.com/sql/%7D> (besucht am 11.08.2023).
- [6] „SQLite“. (2023), Adresse: <https://www.sqlite.org/index.html%7D> (besucht am 11.08.2023).
- [7] H. C. Fortna, „Avionics Integrity Program, Technical Report ASD-TR-84-5030“, Wright Patterson Air Force Base, Techn. Ber., 1984.