PEPPERL+FUCHS

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

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

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

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

P+F Pepperl+Fuchs

MTBF Mean Time Between Failures

HMI Human-Machine-Interface

SQL Structured Query Language

SEQUEL Structured Englisch 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 Radioreperaturwekstadt, 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.

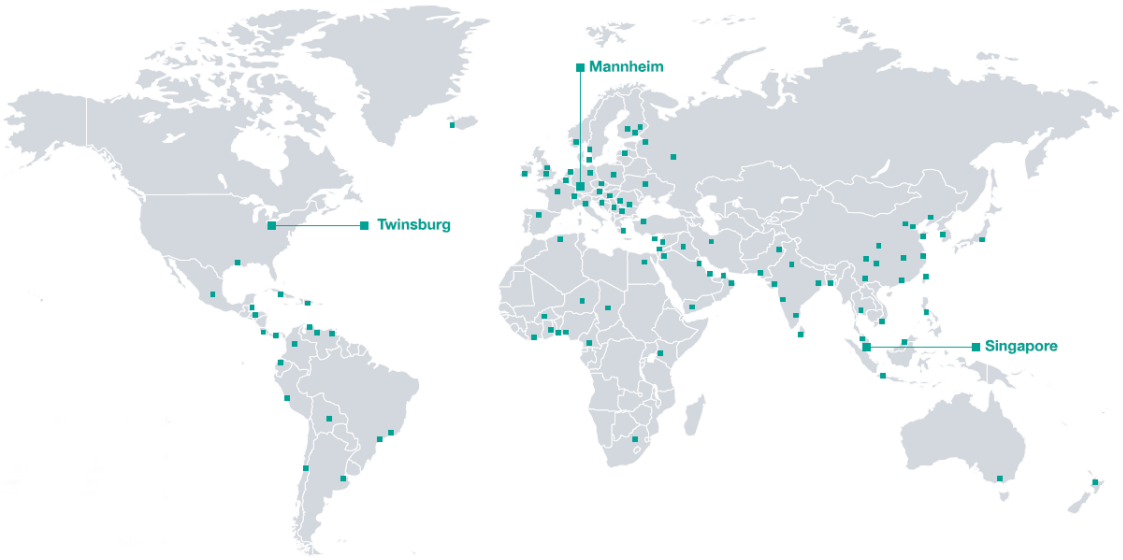

Abbildung 1.1: Standorte der Pepperl+Fuchs SE

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

mehr auf HMI eingehen, warum braucht man diese, 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 das das Programm Skalierbar, Effizient, Robust und gut zu warten ist.
Hierbei bieten sogenannte Design Patterns abhilfe. *Jedes Muster beschreibt zunächst ein in userer Umwelt immer wieder auftretendes Problem, beschreibt dann den Kern der Lösung dieses Problems, und zwar so dass man diese Lösung milionenfach anwendden 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 dessen wird leichter. Zudem wird auch die Implementierung neuer Erweiterungen und das Eindenken in die Software durch gängige Designpatterns 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 Schnitstelle einer Klasse an eine andere von dem Client erwarteten Schnitstelle. Somit ermöglicht das Pattern die Zusammenarbeit von zwei Klassen, welche auf grund ihrer Schnellen nicht möglich wäre. Das Adapter Patern 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 Schnitstelle der klasse nicht mit den aktuellen Anforderungen des clients übereinstimmt. Desweiteren wird das Dattern verwendet, wenn eine wiederverwendbare Klasse erzeugt werden soll, welche mit unabhängigen und nocht vorhersehbaren Klassen interagieren soll.

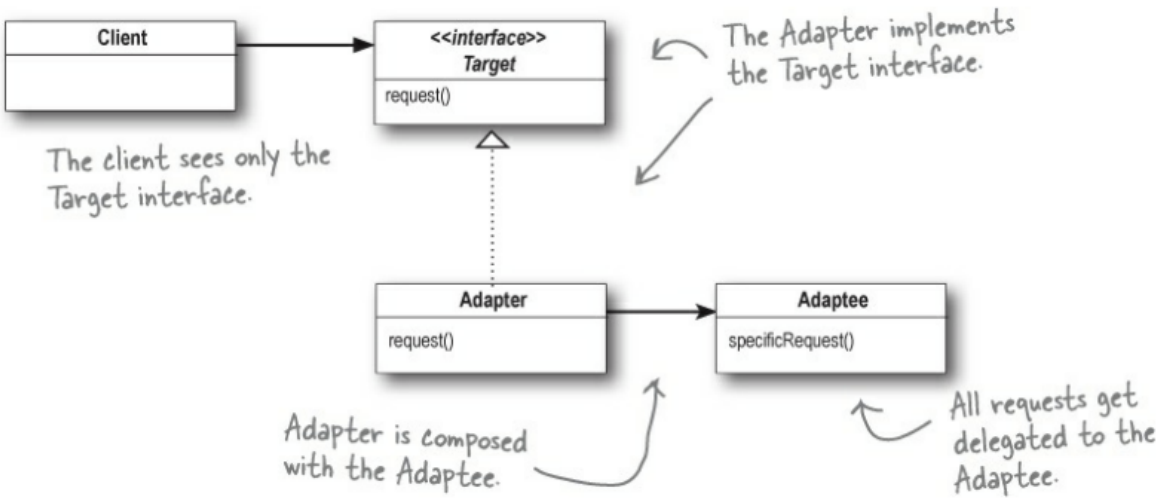

Abbildung 2.1: Adapter Pattern Struktur [1]

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 Patern ermöglicht eine variable und vom Client unabhöngige nutzung des Algorythmus.
Das Pattern kommt zum einsatz wenn eine Reihe von zusammenhängenden Klassen sich nur in Ihrem verhalten unterscheiden, verschiedene varianten eines Algorythmus erfordert werden, der Client keine Kenntnis von den vom Algorythmus verwendeten Daten haben soll, oder eine Klasse verschiedene Verhaltensweisen aufweist.

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

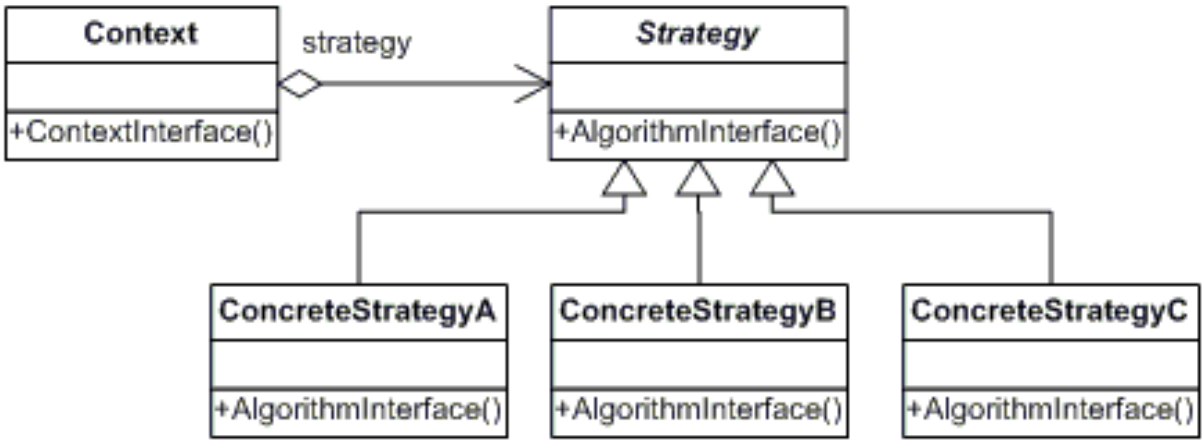

Abbildung 2.2: Strategie Pattern Struktur [1]

dem *Context*, welcher mit eier *ConcreteStrategy* ausgestattet wird. Desweiteren bestitzt 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. Relatione Datenbank und "Nicht relatione Datenbanken". Unterschiede der Datenbankarten machen sich in der Sprache zum Auswerten der DB, ihrer Skalierbarkeit, der Struktur, der Eigenschaften und der Unterstützung durch die Comunity bemerkbar.

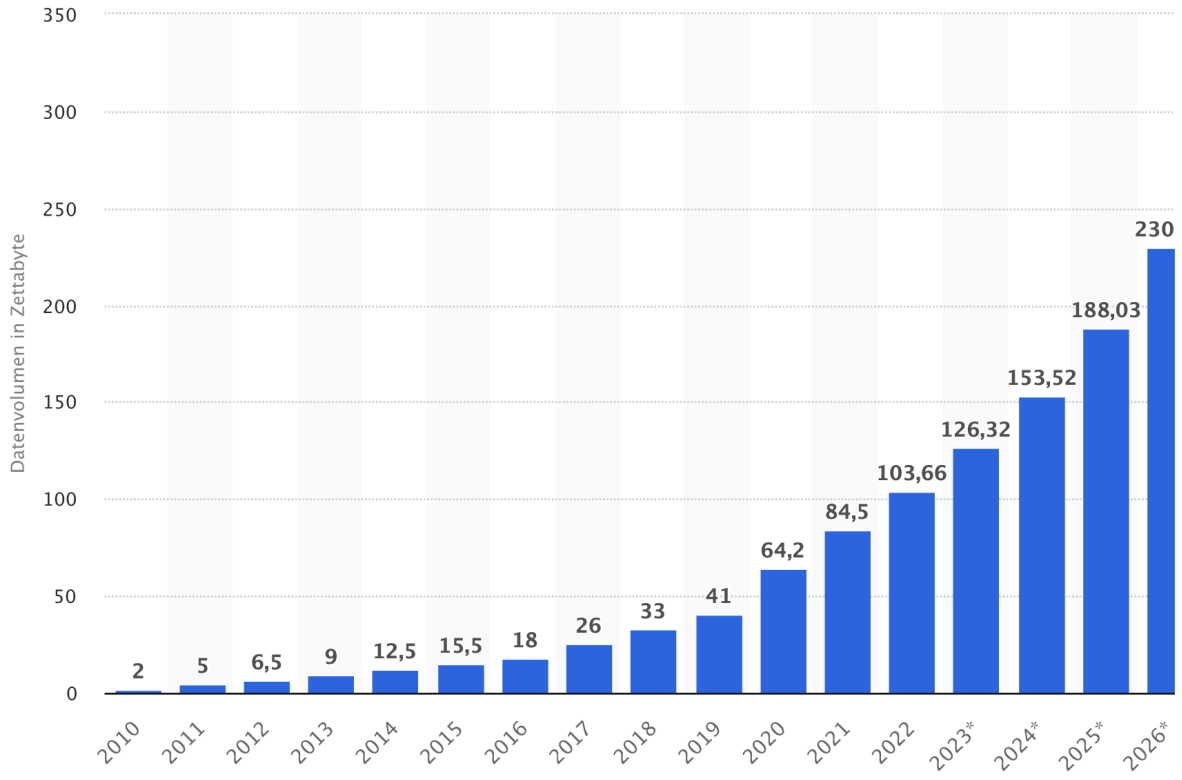

Abbildung 2.3: Volumen der weltweit generierten Daten bis 2027 [3]

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 Englisch Query Language (SEQUEL). Codds Modell für bassiert 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 bespielsweise ein echter Name, eine Telefonnummer und eine Email-Addresse zugewiesen werden.
Die besondere eigenschaft von SQL ist das Konzepte von Arrays. Relationale Datenbanken bestehen aus Arrays, welche sich mit Hilfe von verschiedenen Befehen erzeugen und bearbeiten. **sql**
SQL beitet 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 *DELEAT*, 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 berarbeiten kommen die Befehle *CREATE* und *DROP* zum einsatz. [4]
Natürlich bietet die Programmiersprache eine weit aus komplexere Sysntax, um datensätze sortiet auswerten zu könen. 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 die System.Data.SQLite Bibliothek für C# verwendet.

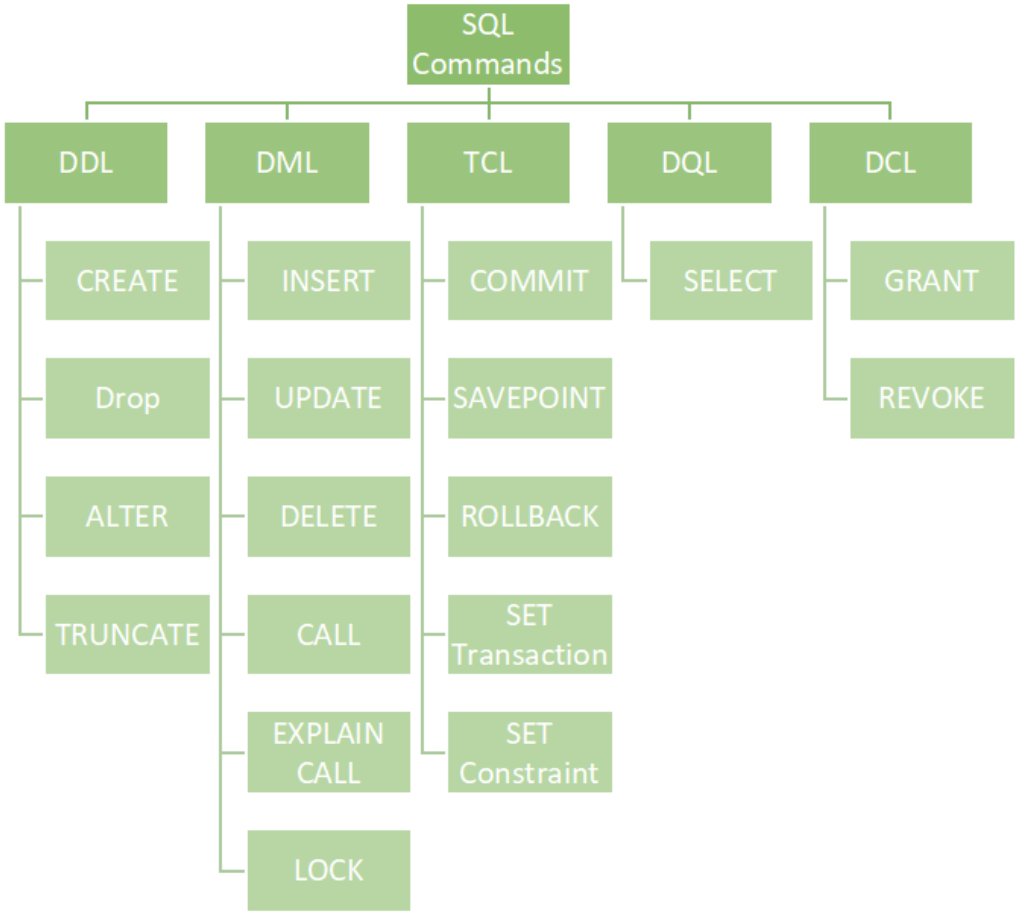

Abbildung 2.4: SQL Befehls Kategorien [4]

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. Wärend sich dieser Zyklus wiederholt können Risse in den Verbindungen entstehen und ausbreiten. Diese können anschließend zu einem Bruch im elektrischen Stromkreis führen. [8]
Die in Abbildung 2.5 abgebildete Bathtub-Kurve ist ein Konzept, welches zur Beschreibung der Lebensdauer von Elektronischer komponenten verwendet wird. Dabei kann die Lebenszeit in drei Abschnitte unterteilt werden. Die Bathtub-Kurve beschreibt eine mittlere Betriebsdauer zwischen Ausfällen. Sie weist drei betriebs phasen auf. In der erste Phase, bekannt als *Infant Mortality*, kommt es durch Konstruktions-, Produktions- und Werkstoffmängel häufig gleich zu begin des Betriebs zu Fehlern und Ausfällen. Geräte die von diesen Problemen nicht betroffen sind, laufen meist zuverlässig durch die Zweite Pahse der Kurve, bekannt als *Random Failures*. Hierbei kommt es nur deutlich seltener und vereinzelt zu Ausfällen. Zum ende der Lebensdauer kommt es, in der *Wear-Out* Phase, durch Alterung und Verschleiß wieder vermehrt zu Ausfällen. [8]

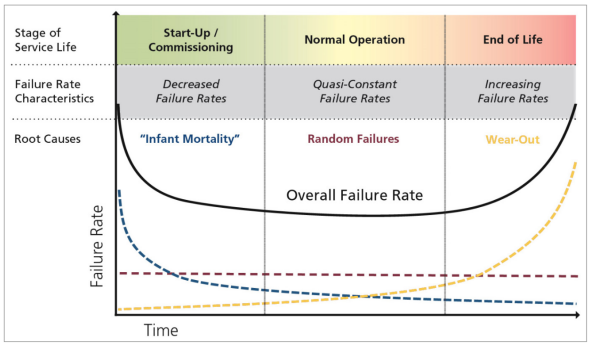

Abbildung 2.5: Bathtub Curve [8]

Der Mean Time Between Failures (MTBF) ist dabei eine statistische Kennzahl, die den durchschnittlichen Zeitraum in Stunden angibt, der zwischen zwei aufeinanderfolgenden Ausfällen einer bestimmten Komponente, eines Systems oder eines Produkts verstrichen is. Dieser weist zudem eine Temperaturabhängigkeit auf. Beispielsweise bei Kapazitäten kann im durchschnitt gesagt werden: *Eine erhähung der Betriebstemperatur um 10°C,*

führt zu einer Halbierung der Lebenserwartung. Ein MTBF von 100h sagt also aus, das ein System im Durchschnitt, 100h laufen wird bevor es zu einem Fehler kommen wird.[9] Die zuverlässigkeit (Reliability) eines Gerätes hingegen ist als die Wahrscheinlichkeit definiert, mit der ein System seine beabsichtigten Funktionen für einen Festgelegten zeitraum erfüllen wird. Hat ein System bei 100h eine Zuverlässigkeit von 0.8, so besteht eine 80% Wahrscheinlichkeit das das System nach 100h noch funktioniert.[9]
Die Zuverlässigkeit eines Systems kann mit formel 2.1 berechnet werden.

$$R(t) = e^{-\frac{t}{\text{mtbf}}} \tag{2.1}$$

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 Ermittelung des Health Status

3.2.3 Ermittelung der Systemstatus Historie

3.2.4 Ermittelung 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: %7Bhttps://datascientest.com/de/sql-alles-uber-die-datenbanksprache#:~:text=SQL%20oder%20%E2%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: %7Bhttps://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/%7D (besucht am 11. 08. 2023).

[5] „SQL Tutorial". (2023), Adresse: %7Bhhttps://www.w3schools.com/sql/%7D (besucht am 11. 08. 2023).

[6] „SQLite". (2023), Adresse: %7Bhttps://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.

[8] E. D. A. AMIR RUBIN, „LIFE EXPECTANCY OF ELECTRONIC EQUIPMENT POST-LOSS", *AREPA*, 2020.

[9] „Understanding MTBF and Reliability". (2020), Adresse: %7Bhttps://relyence.com/2020/05/27/understanding-mtbf-reliability/#:~:text=The%20key%20difference%20is%20that,functioning%20at%20a%20certain%20time.&text=In%20this%20equation%3A,that%20you%20are%20interested%20in%7D (besucht am 14. 08. 2023).