FH Westküste
Wirtschaft und Technik

# Datenverarbeitung und Visualisierung von Umrichter-Testbench-Daten

**Jon Feddersen**
22122017

Bachelorarbeit im Studiengang Elektrotechnik und

Informationstechnik bei

Prof. Dr. rer. nat. Kristina Schädler

Semesteranschrift
Ochsendrift 31
25853 Drelsdorf

Studienfach
Elektrotechnik und
Informationstechnik

Abgabetermin: 01.10.2025

Fachsemesterzahl: 9

## Sperrvermerk

Diese Arbeit enthält vertrauliche Daten und Informationen des Unternehmens, in dem die Bachelor-/Masterarbeit angefertigt wurde. Sie darf Dritten deshalb nicht zugänglich gemacht werden.

Die für die Prüfung notwendigen Exemplare verbleiben beim Prüfungsamt und beim betreuenden Hochschullehrer.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

# 1 Einleitung

Das Aufbereiten von Daten, sowohl in Bezug auf die Struktur, in der die Datensätze gespeichert werden, als auch in der visuellen Darstellung einzelner Datenreihen, ist für das Verstehen der dahinterliegenden Prozesse und der effizienten Arbeit mit ihnen unerlässlich. Das Erfassen von Messdaten und ihre Aufbereitung, so wie ihre Interpretation ist aus der modernen Welt nicht mehr wegzudenken. In allen Bereichen der Wissenschaft werden Messdaten erhoben, egal ob bei medizinischen Studien, Wetterdaten oder in der Industrie erhobenen Testdaten, Sie müssen alle verständlich aufbereitet werden.

Die Speicherung,Aufbereitung und Visualisierung von größeren Datenmengen wir heutzutage meisten mit Softwaretools durchgeführt. Aus diesem Grund gibt es eine Vielzahl von Anbietern die Lizenzen für solchen Softwaretools vertreiben. Diese Tools sind oft aber allgemein gehalten und biete daher für fachspezifische Bereiche nicht den passenden Aufbau und Funktionen, welches sich Effiziente, Umgänglichkeit und vor alledem in der Genauigkeit widerspiegelt.

Das Ziel dieser Arbeit ist es, eine Web-Applikation zu entwickeln, die als solch ein Softwaretool fungieren soll, welches spezifisch für einen Anwendungsbereich zugeschnitten ist. Die Applikation soll XML-Datensätze, die aus einem Umrichter-Teststand stammen strukturiert Speicher und visuell Darstellen kommen. Diese visuellen Daten sollen sowohl firmenintern zur Analyse als auch als in einem Kundenbericht nach außen weitergegeben werden, daher soll die Visualisierung sowohl einfach verständlich also auch professionell gehalten sein, hierbei soll möglichst auf eine intuitive und simple Nutzeroberfläche geachtet werden. Weiterhin soll die Applikation gut erweiterbar sein, um zukünftige Funktionen unkompliziert in die Software einzubetten. Zudem soll eine möglichst einfache Implementierung in die bestehende Softwareumgebung der Firma in dessen Rahmen diese Arbeite durchgeführt wird gewährleistet sein.

Um das gesetzt Ziel und die Anforderungen im Zeitrahmen dieser Arbeit bestmöglich zu erfüllen, wird nach dem Erlangen und Ausformulieren der theoretischen Grundlagen der Erstellung für den eigentlichen Entwicklungsprozess der Web-Applikation eine inkrementelle und iterative Vorgehensweise vorgesehen. Diese soll für Flexibilität und gutes Risikomanagement in Bezug auf die Abgabe sorgen, da selbst bei Komplikation eine funktionelle Version zur Abgabe bereitsteht. Dieser Ansatz zur Entwicklung der Software wird sich nur geringfügig auf den Aufbaue der wissenschaftlichen Arbeit auswirken, sie wird nur in den theoretischen Grundlagen und im Fazit behandelt werden.

Der allgemeine Aufbau des Hauptteils der Arbeit beginnt mit den Kapitel **??**. Grundlagen, die sich mit den theoretischen Hintergründen zur Entwicklung der Applikation, sowie mit den einzelnen zu implementierenden Funktionen beschäftigt. Gefolgt von Kapitel **??**. Analyse und Konzeption, welches sich mit dem bestehenden System und den einzufügenden Strukturen befasst. Das dritte Kapitel **??**. Implemetierung beschreibt die Entwicklung der Web-Applikation. Kapitel **??**. Integration und Test beschreibt die Imtegration in das bestehende System und das Testen der Funktionalität. Abschließend Kapitel **??**. Fazit und Ausblick werden die Ergebnisse zusammengefasst und ein Ausblick auf mögliche Erweiterungen gegeben.

# 2 Definition der Anforderungen und ihre Interpretation

In diesem Kapitel werden die theoretischen Grundlagen für das Entwickeln der Software, so wie das notwendige Verständnis des Teststandes und seine Abläufe behandelt.

## 2.1 Defintion der Anforderungen

In diesem Unterkapitel sollen die gestellten Anforderungen der verschieden Pathein erläutert werden, diese werden in technische und Nutzer-anforderungen aufgeteilt. Die technischen Anforderungen werden den Rahmen für die verwendete Hard- und Software bestimmen. Die Nutzeranforderungen beschreiben die Anforderungen der Verwenderinnen der Apploikation, so wie der Teststand bedienenden Personen.

### 2.1.1 Technische Anforderungen

Die technischen Anforderungen wurden in einem Gesprech mit den nutzenden und leiteten Personen ausgearbeitet.

1. Die Web-Applikation soll auf einem Apache2 Server der Firma laufen.
2. Es wurde festgelegt, das für die Frontendentwicklung bzw. das Erstellen die grafishen Oberfläche mit HTML und CSS durchgeführt werden soll.
3. Das Backend soll mit Python programmiert werde.
4. Für die Visualisierung der Daten als Diagramme soll JavaScirpt genutzt werden.
5. Als Webframework soll Flask verwendet werden, da es sehr minimalistisch und Python basiert ist.
6. Bezüglich der Geschwindigkeit und Leistungsstärke würden keine genauen Anforderungen definiert, die Applikation möglichst effizient Arbeiten, es muss jedoch keine aktive Optimierung erfolgen so lange der Arbeitsfluss nicht stark behindert wird.

### 2.1.2 Nutzeranforderungen

Diese Anforderungen wurden zusammen mit den nutzenden Ingenieur/-in festgelegt.

ausgearbeitet.

1. Die Navigation zwischen den einzellenen Funktionen der Web-Applikation soll über eine Menüleiste erfolgen.
2. Das Einlesen soll möglichst flexibel in Bezug auf die Art des einlesens sein, da sowohl alte Berichte als auch neue **XML!** (**XML!**)-Strukturen aus dem Teststand eingelesen werden können sollen. Zu einem späteren Zeitpunkt können weiter

3. Die eingelesenen Testberichte sollen tabellarisch Ausgegeben werden. Die Tabelle soll nur wichtige Grundinformationen enthalten.
4. Es soll Filterfunktionen für die Bericht-Tabelle zur verfügung gestellte werden.
5. Die zu erstellen Graphen sollen nach den Modulen geordet sein und eine klare Beschrieftung haben.
6. Das Aussehen der Graphen soll sich an den Graphen des Testandprogrammes orintieren, siehe Abbildung **??**.


Abbildung 1: Beispiel der Graphen aus Teststandprogramm
Quelle: Eingenaufnahme

### 2.1.3 Anforderungen an die Datenbank

Diese Anforderungen wurden im Vorfeld mit den späteren Nutzerinnen der Applikation und den potenziellen Verwendern der Datenbank festgelegt.

1. Für die Erstellung der Datenbank soll das Datenbankmanagementsystem MariaDB genutzt werden.

2. Die Datenbank muss in der Lage sein, alle Daten, die in den Berichten vorkommen, zu speichern. Es müsste theoretisch möglich sein, den Bericht nachzukonstruieren.
3. Datendopplungen sollen vermieden werden.
4. Die Tabellen sollen grundlegend die **XML!**-Struktur widerspiegeln, jedoch dürfen häufig vorkommende Daten ausgelagert werden.
5. Für die DUT-Typen und Seriennummern soll jeweils eine separate Informationstabelle erstellt werden.
6. Bei der Erstellung muss berücksichtigt werden, dass es zu einem späteren Zeitpunkt noch weitere Teststände und Testmodule geben kann.
7. Bei der Erstellung soll berücksichtigt werden, dass die Daten zu einem späteren Zeitpunkt für die Analyse benutzt werden sollen.

# 3 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für das Entwickeln der Software, so wie das notwendige Verständnis des Teststandes und seine Abläufe behandelt.

## 3.1 Überblick über den Umrichter-Prüfstand

In diesem Abschnitt wird der Umrichter-Teststand, von dem die zu verarbeitenden Datensätze stammen, beschrieben, da dies für das generelle Verständnis der einzulesenden Datensatzstruktur unerlässlich ist. Die genaue Bezeichnung des Teststandes „USTB DWT Test Bench (XCT0006-1)“ wird im Folgenden als Test-Bench oder Teststand benannt. Diese Art Test-Bench wird im Allgemeinen für die End-of-Line-Prüfung von unterschiedlichen Umrichtern nach ihrer Herstellung genutzt, um die Produktqualität und -funktionalität sicherzustellen. **Main_Manuel_USTB2018**

In dem hier vorliegenden Fall wird der Teststand verwendet, um die aus dem Feld kommenden Umrichter auf ihre weitere Nutzungstauglichkeit zu testen. Die weitere Nutzungstauglichkeit wird ermittelt, indem die Messwerte mit Mittelwerten, die von mehreren fabrikneuen Umrichtern stammen, verglichen werden. Diese Messwerte müssen sich in einem vorher definierten Toleranzbereich befinden, um weiter im Feld verwendet zu werden.

Die Umrichter werden in der gegebenen Fachliteratur zur Test-Bench als **DUTs!** (**DUTs!**) bezeichnet. Diese Bezeichnung kommt auch in den Berichten auf dem Teststand vor, daher hat der Autor diese Abkürzung übernommen.

### 3.1.1 Aufbau des Teststandes

Die Test-Bench besteht aus mehreren Komponenten, die sich im Testraum in unterschiedlichen Schaltschränken befinden. Der Teststand besteht aus folgenden Hauptkomponenten: Die Test-Bench besteht aus mehreren Komponenten, die sich im Testraum in unterschiedlichen Schaltschränken befinden, und der Teststand umfasst folgende Hauptkomponenten:

- Das Netzteil wandelt die 400-V-Netzspannung in eine isolierte Gleichspannung für den Zwischenkreis um. Das Netzteil liefert maximal 80 kW mit 1200 V DC oder 800 V DC, welche Werte verwendet werden, kann vor Teststart bestimmt werden. In Abbildung **??** wird das Netzteil als PSU bezeichnet, was für „Power Supply Unit“ steht.
- Das Elektronik-Rack, auf dem die Mess- und Steuerkomponenten befestigt sind, ist ein wichtiger Bestandteil des Teststandes. Hier befindet sich auch der (XCS2100) System-Controller, der das ganze System mit dem PC, auf dem die Test-Bench-Software läuft, via Ethernet verbindet. In Abbildung **??** mit ER bezeichnet, für „Electronic Rack“.
- Der Testmatrix-Schrank, in dem die Sammelschienen für den Stromanschluss und die Schützen sitzen. In Abbildung **??** mit TM bezeichnet, für „Test Matrix cabine“.

- Der Schrank mit dem Kühlungssystem, da die Umrichter während des Betriebes mit Wasser oder Luft gekühlt werden müssen. In Abbildung **??** mit „Cool1“ bezeichnet.
- Der Carrier, auf dem die Umrichter befestigt werden, ist speziell für bestimmte Umrichter konstruiert. Dieser wird speziell für bestimmte Umrichter konstruiert. In Abbildung **??** wird der Carrier als Carrier1 bezeichnet.

Neben den Hauptkomponenten befinden sich außerhalb des Sicherheitsbereiches, der während des Betriebes nicht betreten werden darf, ein PC mit einer Software zum Steuern der Testeinrichtung sowie eine Betriebsanzeige und ein Notaus. **Main_Manuel_USTB2018**

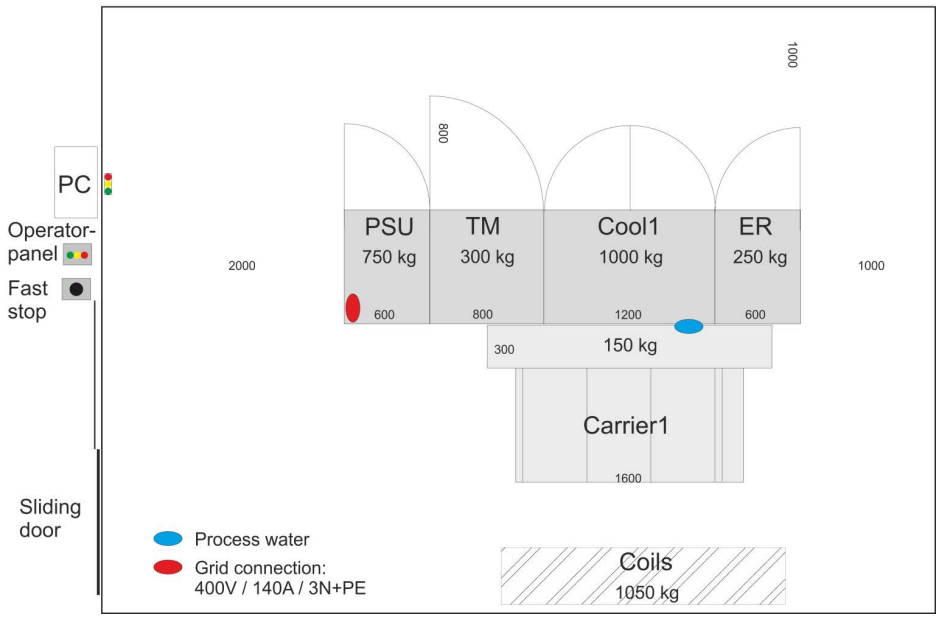


Abbildung 2: Aufbau des Teststandes
Quelle: **Main_Manuel_USTB2018**

### 3.1.2 Testmodule

Es gibt mehrere Testmodule, die auf dem Teststand laufen und verschiedene Funktionen der Umrichter testen. Einige der Funktionen eines DUT können mit dem gleichen Modus eines Testmoduls überprüft werden, indem die entsprechenden Parameter ausgewählt werden. Jeder Testablauf ist autonom und kann mehrmals ausgeführt werden, auch mit unterschiedlichen Parametern.
Im Folgenden wird eine kurze Beschreibung der Funktionen der für diese Arbeit relevanten Testmodule gegeben.

Driver Consumption Test: Der Driver Consumption Test überprüft den Stromverbrauch des Treibers im Leerlauf und während Pulssprüngen.

Pulse Test: Der Impulstest verfügt über drei Funktionsmodi.

- Im Modus **FSW!** (**FSW!**) kann überprüft werden, ob die Halbleiter, die sich in den **DUTs!** befinden, generell schalten.

- Im Modus **OCP!** (**OCP!**) kann die Überstromüberwachung weiche Kurzschlüsse überprüfen. Ein weicher Kurzschluss ist, wenn der Stromfluss nicht sofort und vollkommen unterbrochen wird.
- Im Modus **DSCP!** (**DSCP!**) wird ein harter Kurzschluss, also ein vollständiger und sofortiger Kurzschluss, überprüft.

Power Test: Mit diesem Test werden zwei verschiedene Funktionen getestet werden:

- **BIT!** (**BIT!**) dient dazu, die **DUTs!** zyklisch zu betreiben und so reale Betriebszustände zu simulieren. Zudem kann mit Hilfe dieser Funktion die Kühltemperatur überprüft werden, um die korrekte Wärmeübertragung der Halbleiter sicherzustellen.
- Während des **OTP!** (**OTP!**) wird ein DUT, ähnlich wie beim **BIT!**, nur mit reduzierter Kühlung betrieben, bis die maximal zulässige Kühlkörpertemperatur erreicht ist und die Temperaturschutzschaltung auslöst.**Main_Manuel_USTB2018**

### 3.1.3 Testablauf

Die Schrittfolge des Testablaufes mit Montage der **DUTs!** ist klar festgelegt und vor jedem Durchlauf gleich. Die Montage läuft wie folgt ab:

1. Die Umrichter werden auf dem Carrier befestigt. Es werden meist 3 dieser Geräte gleichzeitig getestet, Abweichungen je nach Bauform. Die Reihenfolge der elektrischen Phasen ist bei Draufsicht des Carriers von links nach rechts U-V-W. Dies ist für das Layout des Berichtes relevant.
2. Der Kühlkreislauf wird angeschlossen, je nach Umrichtertyp Lüftungs- oder Wasserkühlung.
3. Die **DUTs!** werden mit den AC- und DC-Link-Kontakten verbunden, um das Gerät zu betreiben und die DC-Spannung wieder abzuleiten.
4. Das Anbringen von Signalkabeln zwischen DUT und Merkurbox. Die Merkurbox dient als Schnittstelle zwischen **DUTs!** und Testbench.
5. Das Montieren von VCE-Klemmen an die AC-Kontakte der **DUTs!**.

Vor Beginn des Testablaufs benötigt die Test-Bench noch weitere Informationen zu den **DUTs!**, Testsequenzen und der Hardware-Topologie. Diese Daten können über das Scannen von vorliegenden QR- oder Barcodes eingefügt werden. Alternativ können diese auch über ein Eingabefenster im Teststandprogramm eingetragen werden. Je nach Topologiekonfiguration kann auch ein zweiter Träger mit **DUTs!** gescannt werden, Dies kommt bei dem Test, den das Unternehmen durchführt, nicht vor. **Main_Manuel_USTB2018**

Das eigentliche Testen mit den verschiedenen Testmodulen wird autonom durchgeführt. Im Unternehmen läuft der Test regulär wie folgt ab:

1. Durchführung des Driver Consumption Testes, um die Stromversorgung der Treiberplatine auf den Umrichter zu testen.
2. Durchführung des Pulse Testes im Modus Functional-Switching, dies prüft die Umrichter.
3. Durchführung des Power Testes, in den Berichten auch XPower Test genannt. Dieser Test simuliert die Belastung der Umrichter im Feld und überprüft, ob sich die Werte in einem bestimmten Toleranzbereich befinden.

Nach dem Durchlaufen eines Tests wird automatisch ein **XML!**-Datenfile mit den erhobenen Messdaten generiert, die enthaltenen Messwerte und alle vorher bestimmten Einstellungen erstellt und in eine Datei eingefügt.

Bei der Demontage nach dem Testdurchlauf werden alle Schritte in umgekehrter Reihenfolge durchgeführt.

## 3.2 Verarbeitung von XML-Daten

Dieses Kapitel behandelt einige Grundlegende und für dies Arbeit relevante Aspekte von dem Dokumententype **XML!**, da die Messwerte bzw. die Berichte die der Testand generiert im diesem Format vorliegen.

Bei **XML!** handelt es sich um eine Auszeichnungssprache, also eine formale Sprache, die verwendet werden, um die Struktur und Darstellung von Daten oder Texten zu beschreiben **Neumann2019**. **XML!** wurde entwickelt, um Informationen in einem maschinenlesbaren und strukturierten Format zu speichern und zu übermitteln. Sie wird hauptsächlich in Bereichen wie Webdiensten, Datenbanken, Konfigurationsdateien eingesetzt. **XML!** ermöglicht die hierarchische Organisation von Informationen in einem strukturierten Aufbau und kann sowohl für Menschen als auch für Maschinen interpretiert werden.**PeterBrezany2003**

Das Grundkonzept hinter **XML!** war, eine universelle einsetzbare und erweiterbare Sprache zu erschaffen, die von verschiedenen Systemen unabhängig von deren grundlegenden Technologieansatz genutzt werden kann. Hierbei ware das angestrebte Ziel Daten in einem einheitlichen Standard zwischen verschiedenen Anwendungen und Plattformen zu speichern und auszutauschen zu kömmen.**PeterBrezany2003**

### 3.2.1 XML-Strukturaufbau

Eine **XML!**-Datei beginnt mit Prolog, der die **XML!**-Version und die verwendete Zeichencodierung definiert. In Abbildung **??** ist ein häufig genutzter Prolog dargestellt, der auch in den Teststand-Berichten genutzt wird. Die erste Zeile des Prologs ist die sogenannte **XML!**-Deklaration. Die XML-Deklaration enthält häufig die Attribute sind version und encoding, jedoch nur das Attribut version ist Pflicht. Werden auch die anderen notiert, müssen sie in der angegebenen Reihenfolge deklariert werden. Attribut version Mit version wird die verwendete **XML!**-Version angegeben. Das Attribut encoding gibt die im Dokument verwendete Zeichenkodierung an, d. h. mit welcher Codierung die Datei gespeichert wird. Fehlt die Angabe wird als Vorgabe UTF-8 (8-Bit Unicode Transformation Format) verwendet. Neben der **XML!**-Deklaration können im Prolog auch noch Verarbeitungsanweisungen und Verweise auf eine **DTD!** (**DTD!**) deklariert werden, dies sind jedoch optional und für diese Arbeit nicht weiter relevant. **Becher2022**

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Abbildung 3: XML Prolog Beispielcode
Quelle: eigene Darstellung

Der Hauptteil eines **XML!**-Dokument besteht aus einer Reihe von Elementen, die durch Tags markiert sind. Für jedes Element gibt es ein Start- und ein EndTag, welcher das Element beginnt und beendet. Ein Starttag kann beispielsweise „<NamedesTags>“ so aus, dann würde der dazugehörige Endtag „</NamedesTags>“ so aussehen. Der entscheidende Unterschied ist hierbei

der Schrägstrich beim Endtag. Der Name des Elementes wird durch den Inhalt der Keiler- und Großer-Zeichen bestimmt, bei diesem Beispiel wäre der Name „NamedesTags“. Elemente haben einen Inhalt der aus Text, weiteren Elementen oder aus beidem bestehen kann, wenn Elemente andere Elemente beinhalten werden Sie als Elternelemente und die enthaltenen Elemente oft Kindelemente bezeichnet. Diese Eigenschaft der Elementente sorg dafür das **XML!**-Datein einer hierarchischen Baumstruktur folgen. Hierbei wird das oberste Element als Wurzelelement bezeichnet, im Englischen „root element“.**Becher2022**

```
<buch><!-- Das Element "Buch" enth t 2 Kinderelement-->
  <titel>XML-Grundlagen</titel>
  <autor>Max Mustermann</autor>
</buch>
```

Abbildung 4: XML Elemente Beispielcode
Quelle: eigene Darstellung

Jedes Element kann neben Inhalt auch beliebig vielen Attributen ausgestattet sein, die zusätzliche Informationen enthalten. Attribute werden im Start-Tag eines Elements definiert, diese bestehen immer aus einem Attributnamen und einem Wert. Der Wert wird dabei mit Anführungszeichen deklariert, wie in Abbildung **??** gezeigt. **Becher2022**

```
<buch genre="Lehrbuch">ES</buch>
    <!--Hier ist das Attribute "genre" mit dem Inhat "Lehrbuch"-->
```

Abbildung 5: XML Attribute Beispielcode
Quelle: eigene Darstellung

Kommentare werden mit den Tags „<!–“ und „–>“ eingefügt und dienen der Dokumentation oder dem Hinweis auf bestimmte Teile des Codes **Becher2022**. Dies ist für die automatisch generierten Berichte irrelevant, jedoch für das Beschreiben der Beispiele hilfreich. In Abbildung **??** und Abbildung **??** werde diese zur Beschreiben verwendet. Kommentare werden beim Parsen des Dokuments ignoriert, parsen und Paser werden nachfolgenden Kapitel behandelt.

### 3.2.2 Verarbeiten von XML-Dateien

In diesem Abschnitt wird eine Zusammenfassung der grundlegenden Methoden und Techniken gegeben, um **XML!**-Dateien mithilfe verschiedener Tools unabhängig von der verwendeten Programmiersprache zu verarbeiten. Es wird beschrieben, wie man **XML!**-Dateien analysiert, modifiziert und überprüft, um sie für verschiedene Zwecke einsatzbereit zu machen. An Ende des Abschnittes wird jedoch etwas genauer auf Methoden die in Python genutzt werde eingegangen.

**3.2.2.1 XML-Paser** Der erste Schritt beim verarbeiten einer **XML!**-Daten ist das Parsen. Hierbei wird die **XML!**-Datei in ein Programm geladen und in ein Format umgewandelt, das das Programm interpretieren kann. XML-Paser prüfen hierbei auch die **XML!**-Daten auf Korrektheit, also ob die Voralien eingehalten werden und das Dokument vollständig ist. Dabei wird in nicht-validierte Parser und validierende Paser differenziert. Der Unterschied besteht dadrin, dass validierte Paser neben der korrekte Schachtelung und Bezeichnung der Strukturelemente wie die nicht-validierten Paser auch noch auf eine Vorgabe einer Dokumenttypdefinition oder eines Schemas prüfen.**Becher2022**

Paser werden verwendet um eine Applikation über eine **API!** (**API!**) eine Schnittstelle auf ein **XML!**-Dokument zu geben. Bei **API!**s wird in diesem Bereich zwischen zwei Grundtypen unterschieden:**Beche**

Die baumbasierten **API!**s lesen über den **XML!**-Parser das **XML!**-Dokument ein, parst es und erzeugt ein Modell als Baum von Knoten im Arbeitsspeicher. Auf Grunde der im **XML!**-Dokument vorkommenden Informationseinheiten wird in verschiedene Knotentypen unterschieden. Das generierte Modell dient der Applikation für die weitere anwendungsspezifische Verarbeitung. Ein Beispiel für eine baumbasierte **API!**s ist **DOM!** (**DOM!**).

**DOM!**ist Ein objektorientiertes Modell, das die Struktur eines **XML!**-Dokuments abbildet. In der Baumstruktur wird das gesamte Dokument abgebildet, indem jedes Element, Attribut und jeder Text bzw. Inhalt als Knoten gilt. Das **DOM!** hat den Vorteil, dass es das **XML!**-Dokument vollständig im Arbeitsspeicher darstellt, was das Durchsuchen und Bearbeiten des Dokuments erleichtert. Zudem bittet es eine einfache Schnittstelle bereitstellt, um XML-Daten zu erreichen und zu verändern. Allerdings benötigt diese Herangehensweise viel Speicherplatz, da es das gesamte Dokument im Arbeitsspeicher ablegt, was bei großen **XML!**-Dokumenten ein Problem darstellen kann.**Becher2022**

Die ereignisbasierten **API!**s lesen **XML!**-Dokument sequenziell von begin durch ein und meldet während des Lesens jedes Ereignis durch sogenannte Callbacks an die aufrufende Applikation zurück. Ein Ereignis ist ein Signal, das Änderungen in dem Markup-Status anzeigt. Das bedeutet Ereignisse tritten bei Element-Tags, Zeichendaten, Kommentaren, Verarbeitungsanweisungen, sowie bei den Grenzen des Dokumentes auf. Der Parser sendet hierbei durch die Callbacks eine Mitteilung an die aufrufende Applikation, welche Ereignisse eingetreten sind. Das Programm, das den Parser aufgerufen hat, muss nun das Ereignis interpretieren und entsprechend reagieren. Das **XML!**-Dokument wird hierbei nicht vollständig im Arbeitsspeicher gespeichert, sondern in Teilen gelesen und bearbeitet. Deshalb eignen sich ereignisbasierte **API!**s besonders gut für die Verarbeitung großer **XML!**-Dateien, da dies für eine geringe Arbeitsspeicherbelastung sorgt. Trotz der Effizienz hinsichtlich des Speicherverbrauchs sind ereignisbasierte **API!**s für die Datenmanipulation nicht sonderlich gut geeignet. Es wird nämlich durch das Teileweise einlesen keine umfassende Dokumentstruktur wie beim aufgebaut Die baumbasierten **API!**s, welches die Manipulation bzw. bearbeitung erschwert. Ein Beispiel für eine große ereignisbasierte **API!** ist **SAX!** (**SAX!**). Diese **API!** abreitet nach dem oben beschriebenen Prinzip und wurde ursprünglich als Java-**API!** entwickelt, inzwischen gibt es **SAX!** auch für weiter Sprachen wie z. B. C++, Perl und Python.**Becher2022**

**3.2.2.2 Adressierung mit XPath** Neben dem Einlesen der **XML!**-Datei muss zum verwenden der Daten in dem **XML!**-Dokument navigiert werden, um bestimmt Knoten zu adressieren. Aus diesem Grund wurde die Abfragesprache XML Path Language kurz XPath entwickelt. XPath wird hauptsächlich in der Transformationssprache **XSLT!** (**XSLT!**) eingesetzt, welche **XML!**-Dateien in ander Datenformate umwandeln soll. Zudem wird XPath in anderen Programmiersprachen wie z.B. JavaScript, C oder einigen Pythonbibliotheken für die Adressierung von Bestandteilen des **XML!**-Baumes verwendet.

Im Zusammenhang mit **XML!** und XPath wird oft der Begriff Knoten verwindet. Knoten beschreiben Teile des **XML!**-Dokumentes können, aber auch über das Wurzelknoten das gesamte Dokument beschreiben. Knoten können alle im **XML!**-Dokument vorhanden strukturelle Teile sein, daher kann bei Knoten zwischen ein paar unterschiedlichen Typen unterschieden:

- Wurzelknoten, auch Dokumentknoten oder root genannt: Ursprung des **XML!**-Dokuments.
- Elementknoten: ein beliebiges Element.
- Textknoten: ein Text, welcher einem Element untergeordnet ist.
- Attributknoten: ein beliebiges Attribut eines Elements.
- Kommentarknoten: ein beliebiger Kommentar.
- Namensraumknoten: eine beliebige Namensraumangabe eines Elements oder Attributs.
- Verarbeitungsanweisungsknoten: ein XML-Verarbeitungshinweis.

Über dies Knoten kann man durch die **XML!**-Struktur navigieren, aber um nach den Knoten zu adressieren werden Achsen benötigt. Achsen spezifiziert die Beziehung zwischen den Knote, um so die gewünschten Knoten zu selektieren. Es gibt Achsen, welche nur ein Knoten adressieren, so wie Achsen welche mehrere Knoten gleichzeitig auswählen. In der folgenden Tabelle **??** werden die verschiedenen Achsen benannt und beschrieben. Für alle Achsen die keine Kurz-Notation enthalten muss im Code der Name ganz ausgeschrieben werden. **XPath2025**

| **Name** | **Kurz-Notation** | **selektierte Knoten** |
|---|---|---|
| / | / | Wurzelknoten |
| child | (nicht notwendig) | direkt untergeordnete Knoten (Kindknoten) |
| self | . | aktuelle Knoten (Kontextknoten) |
| parent | .. | direkt übergeordneter Knoten (Elternknoten) |
| descendant | .// | alle untergeordnete Knoten |
| descendant-or-self | | alle untergeordnete Knoten sowie der aktuelle Knoten |
| ancestor | | alle übergeordnete Knoten |
| ancestor-or-self | | alle übergeordneten Knoten sowie der aktuelle Knoten |
| following | | alle nachfolgende Knoten (ohne Kindknoten) |
| following-sibling | | alle nachfolgende Knoten (ohne Kindknoten), die den gleichen Elternknoten haben |
| preceding | | alle vorangehende Knoten (ohne alle Elternknoten) |
| preceding-sibling | | alle vorangehende Knoten (ohne alle Elternknoten), die den gleichen Elternknoten haben |
| attribute | @ | Attributknoten |
| namespace | | Namensraumknoten |

Tabelle 1: Übersicht über XPath-Achsen

Quelle: eigene Darstellung nach Tabelle aus **XPath2025**

Um die Selektierung der Knoten noch weiter einzugrenzen, können sogenannte Prädikate verwendet werden. Prädikate befinden sich immer in eckigen Klammer und in diesen können verschiedene Operatoren genutzt werden. Dazu zählen die mathematischen Operatoren +(Addition), -(Subtraktion), *(Multiplikation), div(Division) und mod(Modulo, Divisionsrest), aber auch die logischen Operatoren and (und) und or (oder) sowie die Vergleichsoperatoren =(gleich), !=(ungleich), <(kleiner als), <=(kleiner-gleich), >(größer als) und >=(größer-gleich). Mit diese Operatoren und Aneinanderreihungen von diesen kann eine Vielzahl von Sucheingrenzungen geschaffen werden, welche die Genauigkeit der Suchen stark erhöhen und variable gestalten. **XPath2025**

Ein XPath-Ausdruck, also ein Path welcher zur Adressierung von einem oder mehren Knoten verwendet wird, besteht aus einem oder mehreren Lokalisierungsschritten. EinLokalisierungsschritt besteht aus weiteren 3 Teilen: der Achse, einem Knotentest und bei Bedarf aus einem oder mehreren Prädikaten. Der Knotentest ist hierbei nichts anders als der Name der Knotens. Für die richtige Form des Lokalisierungsschrittes muss der Achsenname mit zwei (Doppel-Doppelpunkt getrennt von dem Knotentest getrennt werden, bei Kurz-Notationen können die Doppelpunkte jedoch weggelassen werden. Wenn anstelle des Knotenteste bzw. dem Knotenname das Wildcard-Zeichen * in einem Lokalisierungsschritte verwendet wird, werden unabhängig vom Namen und Achsen alle Knoten selektiert, welche sich unter dem letzten Lokalisierungsschritte befinden. Die

Lokalisierungsschritte werden mithilfe des Schrägstrichs / voneinander getrennt. **XPath2025** In der folgenden Abbildung **??** ist ein Beispiel für ein XPath-Ausdruck. Dieser XPath würde in einem XML-Dokument mit Bucher das erste Buch auswählen welches vor 1900 erschienen ist, geht dann zu nächsten Geschwisterelement weiter und gibt den Titel zurück.

```
1 //buch[jahr<1900]/following-sibling::buch[1]/titel
```

Abbildung 6: Beispiel XPath-Ausdruck
Quelle: eigene Darstellung

**3.2.2.3 Pythonbibliotheken für XML-Verarbeitung** Für die Bearbeitung von **XML!**-Dokumenten gibt es in Python einige Bibliothek. Diese basieren auf den obengenannten Ansätzen, sind jedoch in dem Umfang deutlich kleiner gehalten und bitten weniger Funktionen. Im folgen wird einige wichtige Bibliotheken genannt und kurz erläuter:

**xml.etree.ElementTree**
Die Standardbibliothek für XML in Python. Sie einfach zu benutzen und es sind keine Zusatzinstallation. Gut geeignet, um kleine bis mittlere XML-Dateien zu verarbeiten. Unterstützt grundlegendes Parsen, Schreiben und einfaches XPath, aber keine **XSLT!** oder komplexe Features für **XML!**-Dokumenten bietet.**ElementTree2025**

**xml.dom.minidom**
Eine DOM-basierte Implementierung in Python. Ermöglicht den Zugriff auf XML über das klassische Document Object Model. Bietet feine Kontrolle, wirkt aber oft sperrig und weniger performant. **xml.dom.minidom**

**xml.sax**
Ein eventbasiertes XML-Parsing. Liest XML zeilenweise und löst Ereignisse aus, wenn Elemente gefunden werden. Sehr speichersparend und ideal für sehr große XML-Dateien, aber auch komplizierter in der Anwendung.**xml.sax**

**lxml**
Die leistungsfähigste XML/HTML-Bibliothek in Python. Basiert auf den C-Bibliotheken libxml2 und libxslt und ist daher sehr schnell. Unterstützt XPath, **XSLT!**, Validierung und kann sowohl sauberes XML als auch „kaputtes“ HTML verarbeiten. Der wird oft als heutiger De-facto-Standard für komplexe XML-Verarbeitung in Python bezeichnet.**lxml**

**BeautifulSoup**
Eigentlich für HTML gedacht, funktioniert aber auch mit XML. Sehr tolerant gegenüber fehlerhaftem Code, was es beim Web-Scraping nützlich macht. Langsamer als lxml, aber sehr einfach in der Benutzung. **BeautifulSoup**

**xmlschema**
Spezialisiert auf **XSD!** (**XSD!**). **XSD!** ist eine Sprache, die die Struktur und den Inhalt von XML-Dokumenten definiert. Xmlschema kann XML-Dokumente validieren und direkt in Python-Objekte

umwandeln. Gut geeignet, wenn du sicherstellen musst, dass XML-Daten einer bestimmten Struktur entsprechen. **xmlschema**

**defusedxml**
Eine sichere Variante der Standardbibliotheken. Schützt vor bekannten XML-Sicherheitslücken wie „Billion Laughs“ oder **XXE!** (**XXE!**). Billion Laughs beschreibt das Einsetzen von stark verschachtelten und sich wiederholenden **XML!**-Strukturen die wenn Sie in den Parser geladen werden dadurch zu Speicher- oder CPU-Überlastung führen. Mit **XXE!** können Befehle im XML-Struktur genutzt werden, welche zu Sicherheitslücke beim XML-Parsing führen. So können Angreifer externe Entitäten einbinden, um Daten auszulesen oder sogar Befehle auszuführen. Wichtig, wenn XML aus unsicheren Quellen kommt. **defusedxml**

**untangle**
Sehr einfache Bibliothek, die XML in Python-Objekte übersetzt. Damit können XML-Strukturen fast wie normale Attribute angesprochen werden. Gut für kleine Projekte, aber eingeschränkt im Funktionsumfang. **untangle**

biblatex

## 3.3 Datenbankentwurf und Normalisierung

Datenbanken sind strukturierte Zusammenstellungen von Daten, die elektronisch gespeichert und verwaltet werden. Ihr Hauptziel ist es, große Datenmengen strukturiert zu speichern, den Zugriff zu optimieren und die Integrität der Daten zu gewährleisten. Datenbanken haben im Vergleich zu anderen Dateisystemen Mechanismen, die mehreren Benutzern die parallele Nutzung ermöglichten, sowie redundante Datenhaltung vermeiden und die effiziente Abfragen über spezielle Sprachen wie die **SQL!** (**SQL!**) ermöglichen.**Fuchs2021**

In der folgen werden die Hauptanforderungen an Datenbanken bzw. Datenbankmanagementsysteme kurz zusammengefasst: **Herrmann2018**,**Fuchs2021**Fuchs2021

- Datenunabhängigkeit: Speicherung unabhängig von Programmen und Plattformen
- Benutzerfreundlichkeit: Einfache Sprachen und grafische Oberflächen
- Mehrfachzugriff: Gleichzeitiger Zugriff für autorisierte Benutzer
- Flexibilität: Wahlfreier Zugriff und fortlaufende Verarbeitung
- Effizienz: Kurze Zeiten für Abfragen, Änderungen und Ergänzungen
- Datenschutz: Zugriff nach Benutzergruppen beschränkt
- Datensicherheit: Schutz vor Fehlern und Ausfällen
- Datenintegrität: Vollständige, korrekte und widerspruchsfreie Speicherung
- Redundanzfreiheit: Daten nur einmal speichern, Redundanz vermeiden

### 3.3.1 Datenbankstruktur

Die Struktur der Datenbank legt fest, wie das Datenbanksystem organisiert ist und wie die Datenelemente angeordnet sind. Das Grundprinzip relationaler Datenbanken sind im Grunde Tabellen und die Beziehungen zwischen diesen. Die Hauptbestandteile sind: **Gadatsch2019**

- Tabellen – das grundlegende Element, welches Daten in Zeilen (Tupeln) und Spalten (Attributen) strukturiert.
- Ein Datensatz in einer Tabelle wird durch Primärschlüssel (Primary Keys) eindeutig identifiziert.
- Schlüssel aus anderen Tabellen (Foreign Keys) Tabellenverknüpfung, um die referenzielle Integrität zu gewährleisten.
- Indizes: Datenstrukturen, die das Beschleunigen von Abfragen ermöglichen.
- Views: sie sind virtuelle Tabellen und beruhen auf den Ergebnissen von Abfragen.

- Constraints: Vorgaben zur Gewährleistung der Datenintegrität (z.B. Wertebereiche, Pflichtfelder oder Dopplungen).

### 3.3.2 Datenintegrität und Normalisierung

Ein methodischer Ansatz zur Reduzierung von Redundanzen und Anomalien ist die Normalisierung. Der Prozess erfolgt schrittweise durch die Normalformen, die durch Indizes (1NF, 2NF, 3NF, BCNF) definiert sind. Alle Normalformen haben spezifische Arten von Datenanomalien zum Ziel: **Gadatsch2019**

- 1. Normalform (1NF): Beseitigung mehrfacher Werte innerhalb einer Zelle.
- 2. Normalform (2NF): Beseitigung partieller Abhängigkeiten.
- 3. Normalform (3NF): Beseitigung transitiver Abhängigkeiten.

Die Gewährleistung, dass Daten korrekt, konsistent und vollständig sind, fällt unter das Konzept der Datenintegrität. Sie wird erzielt durch: **Gadatsch2019**

- Entity-Integrität (eindeutige Primärschlüssel).
- Referentielle Integrität (gültige Fremdschlüsselverweise).
- Domänenintegrität (gültige Wertebereiche und Datentypen).
- Entity-Integrität (eindeutige Primärschlüssel).
- Referentielle Integrität (gültige Fremdschlüsselverweise).
- Integrität der Domäne (gültige Wertebereiche und Datentypen).

### 3.3.3 Vorgehen zur Erstellung der Datenbank

Bei der Erstellung einer Datenbank sollte nach folgendem Schema vorgegangen werden. Bei diesem Schema gilt es, sowohl technische als auch konzeptionelle Aspekte zu berücksichtigen: **Herrmann2018**

1. Anforderungsanalyse

   Zuerst wird im Hinblick auf das Geschäftsumfeld bestimmt, welchen konkreten Zweck die Datenbank erfüllen soll. Hierbei ist die Zweckbestimmung entscheidend, da sie festlegt, welche Daten als relevant gelten. Der Konzeptvorschlag, der das Projektziel definiert und die Vorgehensweise umreißt, ist das Ergebnis.

   - Festlegung der fachlichen und technischen Voraussetzungen.
   - Bestimmung der relevanten Datenquellen und -formate.
   - Definition von Integritäts- und Sicherheitsanforderungen.

2. Konzeptionelles Datenmodell

   Es soll das geschäftliche Umfeld betrachtet werden. Die bestehende Objekte (z.B. Reports mit Materialnummer, Datum, Version), deren Attribute sowie die Beziehungen und Einschränkungen zwischen diesen Objekten. Das Entity-Relationship-Modell (ERM) nach Chen oder das PrecisedERM (PERM) werden häufig zur Modellierung verwendet. Der konzeptionelle Entwurf ist die Grundlage für die nächste Phase und dient als Diskussionsgrundlage.

   - Die Entwicklung eines Entity-Relationship-Modells (ERM), das die reale Welt in Entitäten, Attribute und Beziehungen abbildet.
   - Die Einbeziehung von Kardinalitäten (1:1, 1:n, n:m).

3. Logisches Datenmodell

   Der konzeptionelle Entwurf wird in einen logischen Entwurf umgewandelt, der die fachlichen Konzepte in ein datenbanktechnisches Format überführt. In der Regel kommt das Relationenmodell zum Einsatz, welches die Daten in Form von Tabellen organisiert. Transformationsregeln garantieren, dass Beziehungen und Integritätsbedingungen richtig umgesetzt werden.

   - Transformation des konzeptionellen Modells in ein relationales Schema.
   - Festlegung von Tabellen, Spalten (Attribute), Primärschlüsseln und Fremdschlüsseln.
   - Definition von Datentypen und Zellenkofigurationen (z.B. NOT NULL, UNIQUE, CHECK).

4. Physisches Datenmodell

   Eine physische Datenbankstruktur wird durch **SQL!** erstellt, basierend auf dem logischen Entwurf. Die konkrete Festlegung von Tabellen, Indizes, Constraints usw. erfolgt dabei durch Das System, das wir implementiert haben, wird immer wieder getestet und zusammen mit den Nutzerinnen auf fachliche Richtigkeit überprüft.

   - Realisierung des logischen Modells in einer spezifischen Datenbankmanagementsoftware (z.B. MySQL, Mariadb oder Microsoft SQL Server).er).
   - Die Verbesserung der Speicherstrukturen, der Indexierung und der Partitionierung.

5. Implementierung und Prüfung

   Nach der erfolgreichen Umsetzung wird das System vom Kunden gemäß einem vorher festgelegten Abnahmeplan freigegeben. Ein Wartungsplan kümmert sich anschließend um die Betreuung, schult die Endbenutzer und überwacht die IT-Umgebung kontinuierlich.

   - Den Aufbau der Tabellen und Relationen entsprechend dem Datenbankschema.
   - Die Testdaten implementieren.
   - Die Kontrolle der Funktionalität und Leistung.

## 3.4 Grundlagen der Datenvisualisierung

### 3.4.1 Begriff und Zielsetzung

Unter Datenvisualisierung wird die visuelle Aufbereitung von Daten verstanden, um Muster, Trends, Zusammenhänge oder Anomalien erkennbar zu machen. Sie stellt ein zentrales Hilfsmittel dar, um komplexe Informationen effizient zu kommunizieren und kognitive Verarbeitungsprozesse zu unterstützen[Shneiderman, 1996]. Durch die grafische Darstellung wird es dem Betrachter ermöglicht, große Datenmengen intuitiv zu erfassen, ohne ausschließlich auf numerische oder textuelle Formen angewiesen zu sein. Die Zielsetzung der Datenvisualisierung umfasst in der Regel drei Kernaspekte[Ware, 2021]:

1. Exploration: Unterstützung bei der Entdeckung neuer Zusammenhänge und Hypothesen.
2. Analyse: Erleichterung der detaillierten Untersuchung von Strukturen und Abhängigkeiten.
3. Kommunikation: Vermittlung von Ergebnissen an unterschiedliche Zielgruppen.

### 3.4.2 Theoretische Grundlagen

Die Grundlage wirksamer Datenvisualisierung liegt in der menschlichen Wahrnehmungspsychologie. Insbesondere die Gestaltungsgesetze (z.B. Gesetz der Nähe, Ähnlichkeit und Kontinuität) spielen eine zentrale Rolle, da sie bestimmen, wie Informationen visuell gruppiert und interpretiert werden[Wertheimer, 1923].

Zusätzlich beschreibt die Theorie der *kognitiven Belastung* (Cognitive Load Theory), dass Darstellungen so gestaltet werden sollten, dass sie die Arbeitsgedächtniskapazität nicht überlasten[Sweller, 1988].

Ein weiterer theoretischer Rahmen ist Shneidermans Visual Information-Seeking Mantra:

„Overview first, zoom and filter, then details-on-demand.“

Dieses Prinzip betont die Notwendigkeit einer schrittweisen Annäherung an Daten, um vom Gesamtüberblick bis hin zu spezifischen Details zu gelangen.

### 3.4.3 Formen der Datenvisualisierung

Datenvisualisierungen lassen sich in verschiedene Kategorien einteilen[Few, 2012]:

- Explorative Visualisierung: Interaktive Darstellungen, die zur Untersuchung und Hypothesengenerierung dienen.
- Explikative Visualisierung: Fokussierte, oft statische Darstellungen, die Ergebnisse gezielt kommunizieren.

- Interaktive Dashboards: Kombination mehrerer Visualisierungstypen, häufig für Monitoring und Entscheidungsunterstützung.

Je nach Datentyp und Analyseziel kommen unterschiedliche Diagrammformen und Techniken zum Einsatz:

- Zeitreihen: Liniendiagramme, Flächendiagramme
- Kategorische Daten: Balken- und Säulendiagramme
- Zusammenhänge: Streudiagramme, Blasendiagramme
- Verteilungen: Histogramme, Boxplots
- Hierarchien und Netzwerke: Baumdiagramme, Graphvisualisierungen

### 3.4.4 Qualitätskriterien

Eine gute Datenvisualisierung erfüllt folgende Kriterien[Tufte, 2001]:

- Klarheit: Vermeidung unnötiger grafischer Elemente („Chartjunk“)
- Genauigkeit: Wahrheitsgetreue Darstellung ohne Verzerrung von Skalen oder Proportionen
- Effizienz: Schnelle Erfassbarkeit der relevanten Information
- Ästhetik: Ansprechende Gestaltung zur Förderung der Akzeptanz
- Barrierefreiheit: Berücksichtigung farbsehschwacher Nutzer durch geeignete Farbpaletten

### 3.4.5 Technologische Aspekte

Mit dem Fortschritt moderner IT-Systeme haben sich leistungsfähige Werkzeuge und Bibliotheken für die Datenvisualisierung etabliert. Beispiele sind Matplotlib und Plotly im Python-Umfeld, D3.js im Webbereich.

In Webapplikationen ermöglichen interaktive Bibliotheken eine nahtlose Einbettung in Benutzeroberflächen, wodurch sowohl explorative als auch explikative Ziele unterstützt werden.

### 3.4.6 Graphen erstellung mit JS

#### Chart.js

Eine einfache und beliebte Bibliothek für Standarddiagramme wie Balken, Linien oder Kreisdiagramme. Schnell eingebunden, gute Standardoptik, ideal für kleinere Projekte oder schnelle Visualisierungen.

### D3.js

Eine sehr mächtige Low-Level-Bibliothek, die mit SVG, Canvas und HTML arbeitet. Extrem flexibel für individuelle und interaktive Visualisierungen, aber mit einer steilen Lernkurve.

### Plotly.js

Bietet viele interaktive Diagramme (Zoom, Hover, Export als PNG). Unterstützt auch 3D- und wissenschaftliche Charts. Gut geeignet für Dashboards und Datenanalyse.

### ECharts (Apache ECharts)

Eine leistungsstarke Open-Source-Bibliothek von Apache. Unterstützt viele Diagrammtypen (inkl. Heatmaps, Maps, Candlesticks) mit Animationen und Interaktivität. Besonders beliebt für komplexe Enterprise-Web-Apps.

### Highcharts

Eine professionelle, kommerzielle Lösung (kostenlos für private Nutzung). Sehr einfach konfigurierbar, liefert Business-taugliche Diagramme mit vielen Optionen. Oft in Unternehmen eingesetzt.

### vis.js

Spezialisiert auf Netzwerk- und Zeitachsenvisualisierungen. Eignet sich besonders für Knoten-Graphen, Beziehungen oder Prozessdiagramme mit Interaktivität.

### Recharts

Eine React-spezifische Bibliothek, die auf D3.js basiert. Bietet eine einfache Komponenten-API für gängige Diagramme. Ideal, wenn deine Web-App mit React entwickelt ist.

## 3.5 Anforderungen an modulare Softwareentwicklung

Die modulare Softwareentwicklung ist ein etabliertes Denkmuster, das darauf abzielt, komplexe Softwaresysteme in klar abgegrenzte, wiederverwendbare und unabhängig voneinander entwickelbare Einheiten zu zerlegen. Ziel ist es, sowohl die Wartbarkeit, Erweiterbarkeit als auch die Qualität der Software zu erhöhen. Um diese Ziele zu erreichen, müssen spezifische Anforderungen an die Gestaltung und Umsetzung modularer Systeme beachtet werden.

### 3.5.1 Klare Modulabgrenzung und Verantwortlichkeiten

Ein Modul sollte eine klar definierte Aufgabe erfüllen und über eine eindeutig abgegrenzte Funktionalität verfügen. Diese Trennung wird in der Literatur häufig als *Separation of Concerns* (SoC) bezeichnet. Durch eine eindeutige Abgrenzung lassen sich Abhängigkeiten reduzieren, wodurch Änderungen in einem Modul nur minimale Auswirkungen auf andere Systemkomponenten haben.

### 3.5.2 Geringe Kopplung und hohe Kohäsion

Zwei zentrale Qualitätsmerkmale modularer Systeme sind niedrige Kopplung und hohe Kohäsion.

- Kohäsion beschreibt, wie eng die Elemente eines Moduls zusammenarbeiten, um eine spezifische Aufgabe zu erfüllen.
- Kopplung hingegen beschreibt den Grad der Abhängigkeit zwischen einzelnen Modulen. Ein hoher Kohäsionsgrad bei gleichzeitig niedriger Kopplung erleichtert sowohl die Wiederverwendung als auch die Wartung.

### 3.5.3 Einheitliche Schnittstellen

Module interagieren über klar definierte und stabile Schnittstellen. Diese sollten standardisiert, dokumentiert und möglichst unabhängig von internen Implementierungsdetails sein. Eine wohldefinierte API (Application Programming Interface) ermöglicht die parallele Entwicklung mehrerer Module und erleichtert zukünftige Erweiterungen.

### 3.5.4 Wiederverwendbarkeit

Ein wesentliches Ziel der Modularisierung ist die Wiederverwendbarkeit von Modulen in unterschiedlichen Projekten oder Kontexten. Wiederverwendbare Module reduzieren Entwicklungsaufwand, erhöhen die Konsistenz und tragen zur Qualitätssteigerung bei. Hierfür ist eine generische und konfigurierbare Implementierung erforderlich.

### 3.5.5 Erweiterbarkeit und Anpassungsfähigkeit

Modulare Software muss so konzipiert sein, dass neue Funktionen ohne weitreichende Änderungen am bestehenden System integriert werden können. Dieses Prinzip wird oft mit dem *Open/Closed Principle* aus den SOLID-Prinzipien beschrieben: „Software entities should be open for extension, but closed for modification“.

### 3.5.6 Testbarkeit

Durch die Abgrenzung von Modulen wird eine gezielte und unabhängige Prüfung einzelner Systemkomponenten möglich. Unit-Tests, Integrationstests und Mocking-Strategien profitieren stark von einer modularen Struktur. Testbare Module führen zu einer höheren Softwarequalität und geringeren Fehlerquoten im Betrieb.

### 3.5.7 Konsistenz und Standardisierung

Die Implementierung sollte einheitliche Konventionen in Bezug auf Namensgebung, Dokumentationsstandards und Code-Struktur aufweisen. Konsistenz erleichtert die Zusammenarbeit in Entwicklerteams und minimiert Missverständnisse.

# 4 Analyse und Konzeption

## 4.1 Analyse der generierten XML-Berichte und bestehenden Strukturen

In diesem Unterkapitel wird der Aufbau der automatisch generierten Testberichte aus dem Teststand und der vorhandenen Ausgabe- und Speicherstruktur erläutert. Dies ist relevant für die Erstellung der Datenbank und das Verständnis der Ein- und Auslesefunktionen der Web-Applikation.

### 4.1.1 XML-Berichtsstrukturschema

Die vom Teststand automatisch generierten Berichte folgen einem konstanten Strukturschema, welches sich bis zu einer gewissen Ebene der XM-Struktur in jedem Bericht wiederholt. Wie im Kapitel „Grundlagen“ beschrieben, kann ein Element Attribute, andere Elemente und einen Inhaltswert beinhalten.

Das Stammelement heißt in allen automatisch generierten Berichten „test“ und besitzt immer das Attribut „id“. Die Zahl in diesem Attribut beschreibt den Typ des **DUTs!** die in diesem Test geprüft wurden.

Das Stammelement hat die Unter- bzw. Kinderelemente „info“, „testbench“, „string“ und dreimal „testmodule“. Diese Elemente haben wiederum alle weiter Unterelemente und Attribute. In der folgenden Abbildung **??** ist die unveränderliche Struktur eines Testberichtes abgebildet, welcher erfolgreich durchgeführt wurde.

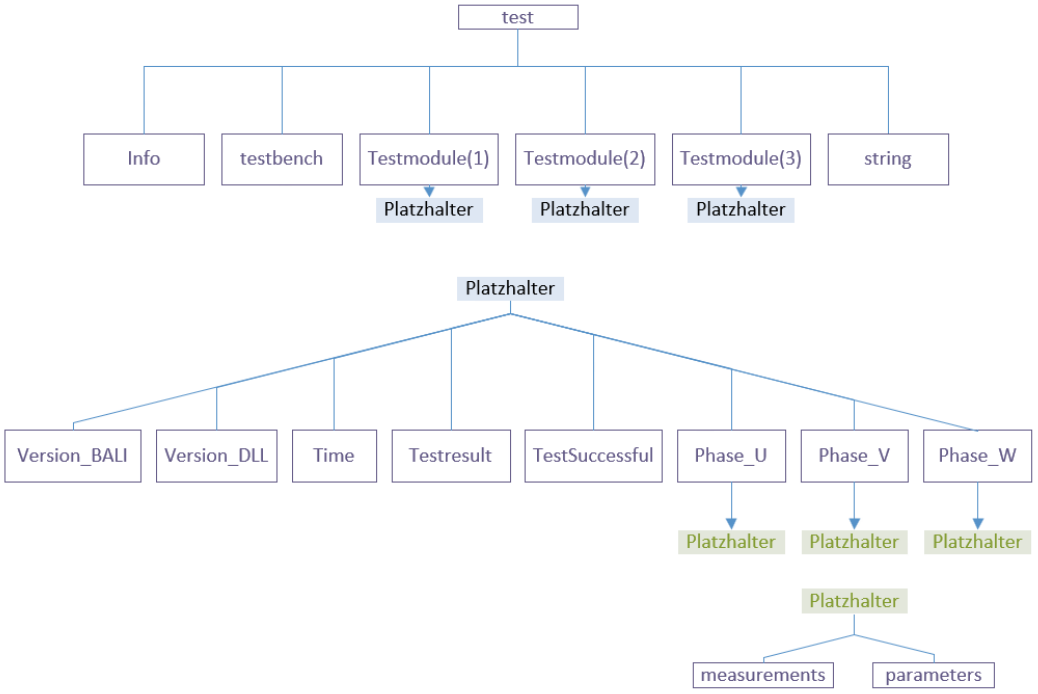


Abbildung 7: Aufbau unveränderlicher XML-Berichtstruktur
Quelle: Eigene Darstellung mit Microsoft Visio

Hierbei sollte angemerkt werden, dass in allen Testberichten die Elemente, die kein Unterelement besitzen gleich aufgebaut sind. Sie besitzen mindestens das Attribut „name“ und ihre Elementenbezeichnung ist immer nach dem Datentyp ihres Inhalts benannt. Für ein besseres Verständnis siehe Abbildung **??**.

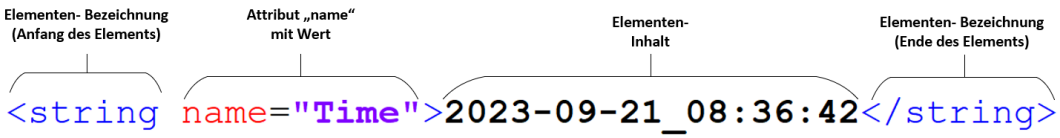


Abbildung 8: Elementaufbau aus XML-Bericht
Quelle: Eigene Darstellung mit Microsoft Visio

Jedes der Elemente „testmodule“ hat ein Attribut „name“, durch welches Sie unterschieden werden können. Das Element „string“ hat ein Namensattribut mit der Bezeichnung „test sequence status“. Der Inhalt dieses Elementes gibt an, ob der Test erfolgreich abgeschlossen wurde oder ob es einen Fehler bzw. eine Grenzüberschreitung von Messwerten gab. Die hier genannten eigenschaften sind in der Abbildung **??** zu erkennen.

```
<?xml version="1.0" encoding="UTF-8"?>
    <test id="124">
    <info></info>
    <testbench></testbench>
    <testmodule name="DriverConsumptionTest"></testmodule>
    <testmodule name="PulseTest␣FSW␣L"></testmodule>
    <testmodule name="XPowerTest"></testmodule>
    <string name="test␣sequence␣status">
        Test sequence successfully finished.</string>
    </test>
```

Abbildung 9: Direkte Kinderelement unter Stammelement
Quelle: Eigene Darstellung

Das Element „info“ enthält in seinen Unterelementen die allgemeinen Testinformationen, wie die Startzeit des Tests, die Typen-ID, die Konfigurationsbezeichnung des Teststandes, die Bezeichnung des angeschlossenen Carriers und die Seriennummern des **DUTs!**. Diese Elemente haben keine Unterelemente und sind alle mit einem Namensattribut und einem Inhalt definiert. Für die genauen Bezeichnungen und Struktur siehe Abbildung **??**

```
<info>
        <string name="Time">2023-09-21_08:36:33</string>
        <string name="Material_number">124</string>
        <string name="Configuration">L_3DUT_3P1Q</string>
        <string name="ID_Carrier_left">Carrier 1</string>
        <string name="ID_DUT_R">10-29550</string>
        <string name="ID_DUT_S">10-29396</string>
        <string name="ID_DUT_T">10-29482</string>
</info>
```

Abbildung 10: XML-Strukturbeispiel Info-Element
Quelle: Eigene Darstellung

Die Teststandbezeichnung und die Versionen der Hard- und Software werden im Element „testbench“ gespeichert. Diese Elemente enthalten auch keine weiteren Unterelemente und sind alle mit einem Namensattribut und einem Inhalt versehen. Die genauen Bezeichnungen und die Struktur sind in Abbildung **??** zu sehen.

```
<testbench>
        <string name="Testbench_name">USTB_DtWind</string>
        <string name="Testbench_HW_revision">V1.5</string>
        <string name="Version_GUI">V1.4.1</string>
        <string name="Version_controller">V1.3.2</string>
</testbench>
```

Abbildung 11: XML-Strukturbeispiel Testbench-Element
Quelle: Eigene Darstellung

Die Testmodule-Elemente sind alle nach dem gleichen Schema aufgebaut. Die ersten Unterelemente enthalten Grundinformationen zu dem Status des Testmoduls, der Tages- und Uhrzeit des Tests und welche Versionen davon verwendet wurden. Danach befinden sich noch drei weitere Elemente im Element „module“. Diese Elemente heißen „Phase U“, „Phase V“ und „Phase W“. Sie besitzen kein Namensattribut, siehe Abblidung **??**.

```
<testmodule name="DriverConsumptionTest">
        <integer name="TestSuccessful" min="1" max="1">1</integer>
        <string name="Testresult">
        Test finished successfully!</string>
        <string name="Time">2023-09-21_08:36:42</string>
        <string name="Version_DLL">V1.2.4</string>
        <string name="Version_BALI">V1.2.4</string>
    <Phase_U>...</Phase_U>
    <Phase_V>...</Phase_V>
    <Phase_W>...</Phase_W>
</testmodule>
```

Abbildung 12: XML-Strukturbeispiel Testmodulheader
Quelle: Eigene Darstellung

Diese Phasenelemente enthalten die Parameter und die Messdaten für die DUTs. Jede Phase enthält die Messwerte für ein DUT. Somit besitzen alle Module unter dem Element „Phase U“ die Messwerte und Parameter für den ersten DUT, dessen Seriennummer unter dem Element mit dem Namensattribut „ID DUT R“ in Abbildung **??** zu finden ist.

In dem Element „parameters“, welches in jedem Phasenelement vorkommt, befinden sich die voreingestellten Rahmbedingungen und Anforderungen für das Testmodul. Diese enthalten alle ein Namensattribut und einen Inhalt.

Die Elemente unter dem Eintrag „measurements“ enthalten die Messdaten der Testmodule. Sie haben oft noch extra Attribute wie „min“ oder „max“, welche die Toleranzgrenzen beschreiben. Diese Toleranzgrenzen zeigen, in welchem Bereich der jeweilige Elementinhalt angesiedelt sein muss, um kein positives bzw. fehlerfreies Ergebnis zu erzeugen. Die meisten Attributswerte finden sich schon in den Parametern wieder und sind somit doppelt im Bericht zu finden. Nur bei den Floatblock-Elementen, also Elementen mit der Bezeichnung Floatblock, sind besondere Attribute zu finden, welche nicht gedoppelt vorkommen. Diese Elemente umfassen zusätzlich die Attribute „size“ und „duration“, welche die Anzahl der in dem Inhalt enthaltenen Floatwerte und die Dauer der Werteaufnahme angeben dies ist in Abbildung **??** zu erkennen. Diese sind für die grafische Darstellung besonders relevant.

```
<floatblock name="Time" size="698" duration="959.952" unit="s">
0.71365,2.1370,3.5323,...</floatblock>
```

Abbildung 13: Beispiel XPowertest Floatblock-Elemente
Quelle: Eigene Darstellung

Auf der Phasenebene treten keine Veränderungen in der **XML!**-Struktur auf, variieren lediglich die Elemente unter den Phasenelementen von Bericht zu Bericht etwas. Die beträchtliche Ausnahme von dieser Regel bilden die Berichte, bei denen ein Fehler aufgetreten ist oder sogar der Testvorgang abgebrochen wurde. Bei diesen Berichten hört die **XML!**-Struktur bei den fehlgeschlagenen

Testmodulen schon auf. Der Testbericht ist aber in sich geschlossen. Das bedeutet, die **XML!**-Struktur ist vollkommen und kann von einem **XML!**-Parser eingelesen werden. Es befinden sich nur weniger Testmodul-Elemente in dem Bericht und im Element für den Testsequenzstatus ist folgender Inhalt enthalten „Test sequence finished with errors“. Zudem wird bei dem Testmodul, bei dem der Fehler aufgetreten ist, unter dem Element mit dem Namen „Testresult“ ein Fehlercode und seine Bedeutung ausgegeben. In Abbildung **??** ist ein Beispiel für einen Bericht, welcher beim ersten Testmodul einen Fehler ausgegeben hatte. Bei einem Ausfall eines der zu testenden **DUTs!** gibt es die Besonderheit, dass die anderen noch funktionsfähigen **DUTs!** keine weiteren Datenaufnahmen mehr machen können. Denn das System benötigt alle drei Phasen, um den Test durchzuführen. Die **XML!**-Struktur ist in sich geschlossen, aber es werden keine Daten von den anderen **DUTs!** mehr aufgenommen.

```
<testmodule name="DriverConsumptionTest">
        <integer name="TestSuccessful" min="1" max="1">0</integer>
        <string name="Testresult">Idrv not within limits on Phase U
0x30010019 !
0x30010013</string>
        <string name="Time">2021-03-10_09:10:37</string>
        <string name="Version_DLL">V1.2.4</string>
    <string name="Version_BALI">V1.2.4</string>
    <Phase_U>...</Phase_U>
    <Phase_V>...</Phase_V>
        <Phase_W>...</Phase_W>
</testmodule>
```

Abbildung 14: Beispiel Fehler bei DriverConsumptionTest
Quelle: Eigene Darstellung

## 4.2 Datenbankdesign und Strukturkonzeption

In diesem Unterkapitel soll der Verlauf der Erstellung der Datenbank beschrieben und nachvollzogen werden. Es wurde in den Anforderungen festgelegt, dass MariaDB zu erstellen genutzt werden soll. MariaDB ist ein kostenloses, relationales Open-Source-Datenbankmanagementsystem, das aus einer Abspaltung von MySQL entwickelt wurde. MySQLs früherer Hauptentwickler Michael Widenius hat das Projekt ins Leben gerufen.

Betrachtung der Anforderungen

Für das genaue Interpretieren und Umsetzen der Anforderungen wurde festgelegt, dass die Ersteller/-in der Datenbank erst ein Konzept nach den festgelegten Anforderungen erarbeiten und umsetzen. Diese Umsetzung wird dann im Laufe der weiteren Erstellung der Applikation, falls nötig, angepasst.

Konzeptionelles Datenmodell

Für das Erstellen der Konzeption des Datenbankmodelles wurde die XML-Struktur analysiert, um die Kenntnisse und Muster aus Unterkapitel **??** abzuleiten. Aus diesen Erkenntnissen wurde dann ein ER-Modell erstellt, siehe Abbildung **??**. Im Folgenden wird das ER-Modell beschrieben.

Durch das Durchführen eines Testes wird ein Bericht erstellt. Dieser Bericht enthält einige Attribute, welche Testinformationen, darunter die Seriennummern, das Datum und die Uhrzeit des Tests, die Materialnummer (DUT-ID), die Konfigurations-Version und die Carrier-Bezeichnung enthalten. Zudem besitzt der Bericht eine Unterentität Testbenchinformationen“, die die Hard- und Softwareversionen des Teststandes enthält. In der Unterentität Testmodelle befinden sich Versions-, Zeit- und Ergebnisinformationen sowie drei Datensätze, welche die Testparameter und die Messdaten enthalten. Diese Datensätze können den Phasen des Stromnetzes des Teststandes und somit den DUTs zugeordnet werden. Zusätzlich zu dem Aufbau wurden hier schon Schlüsselattribute hinzugefügt, die später die primären Schlüssel der Datenbanktabellen dienen sollen, orange markiert. Die Attribute sind hier nicht genau nach dem Inhalt der XML-Dateien benannt aber an ihre genauen Bezeichnungen angelehnt.

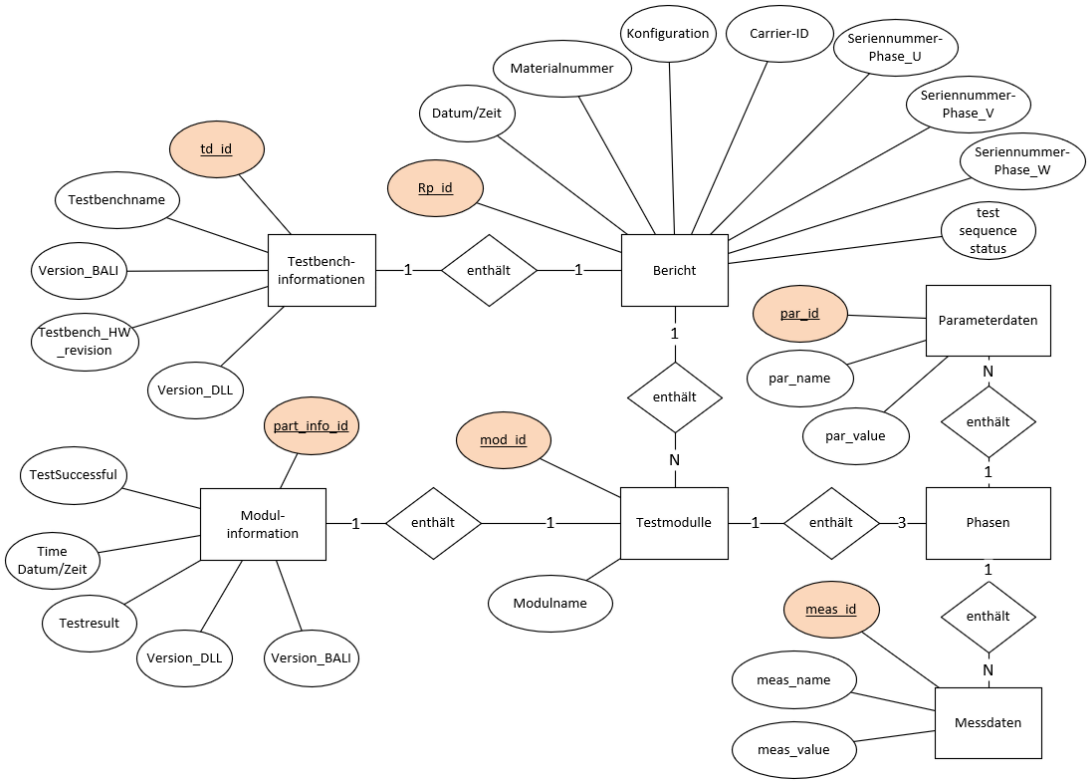


Abbildung 15: ER-Modell Überlegung der Datenbankstruktur
Quelle: Eigene Darstellung mit Microsoft Visio

Aus diesem ER-Model wurde die Datenbanktabelle und ihre Verbindungen über Fremdschlüssel abgeleitet. In Abbildung **??** werden die Datenbanktabellen, ihre genauen Namensbezeichnungen der Tabelle, dem Datentyp in der Datenbank und ihr Inhalt, sowie ihre Verbindungen abgebildet.

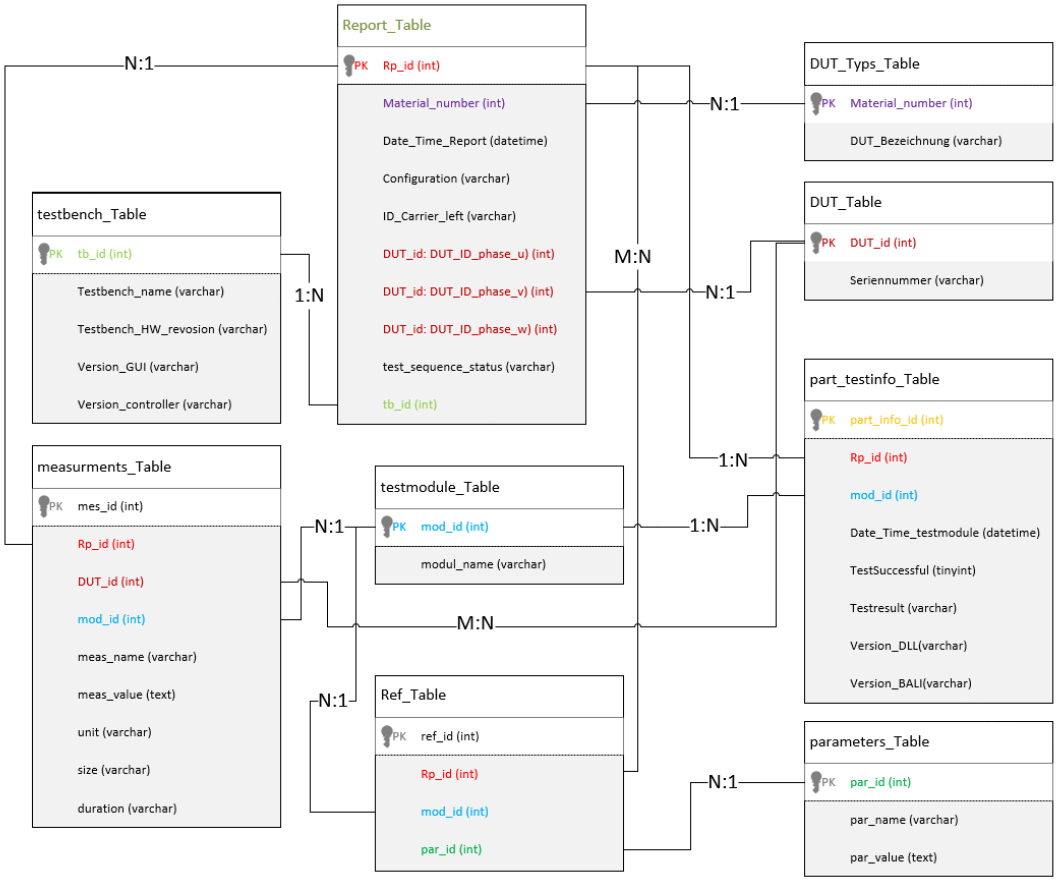


Abbildung 16: Darstellung der Datenbanktabellen und ihrer Verbindungen
Quelle: Eigene Darstellung mit Microsoft Visio

## 4.3 Grundkonzept des Benutzeroberflächen-Design

In diesem Abschnitt soll das Grundkonzept der Benutzeroberfläche beschrieben und dargestellt werden.

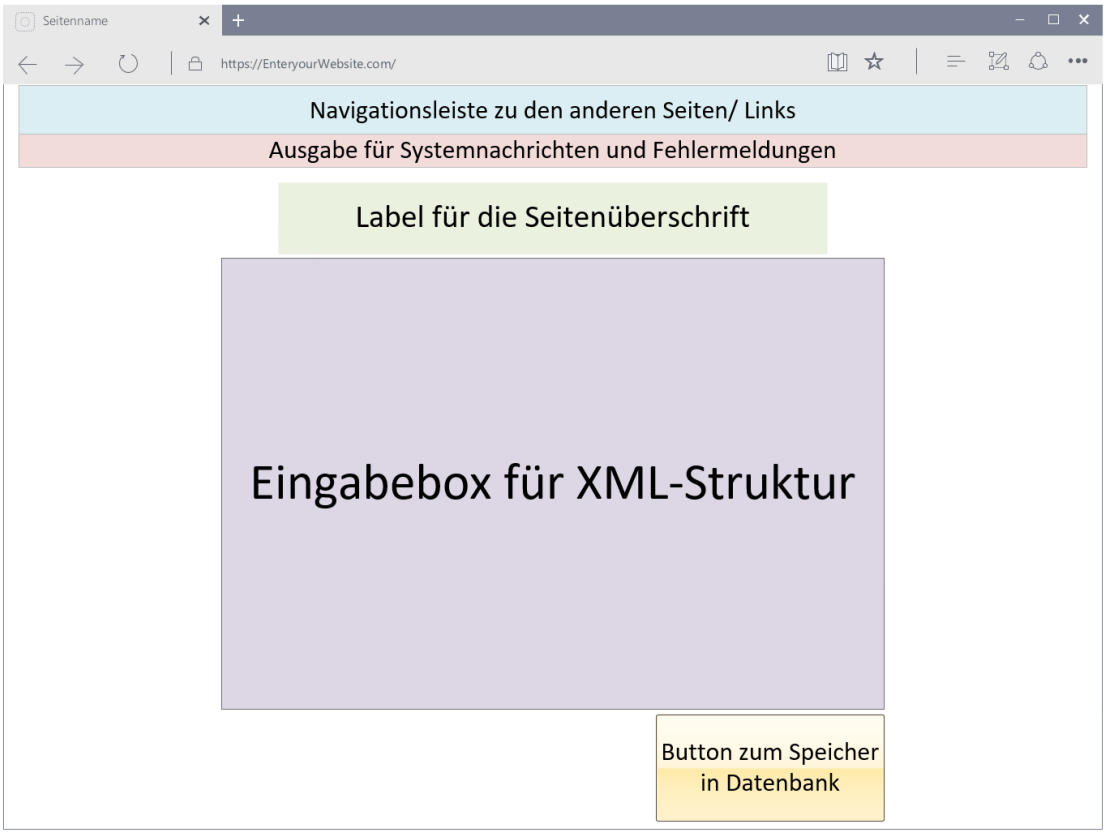


Abbildung 17: Benutzeroberflächenentwurf der Seite zum einlesen der XML-Struktur
Quelle: Eigene Darstellung mit Microsoft Visio

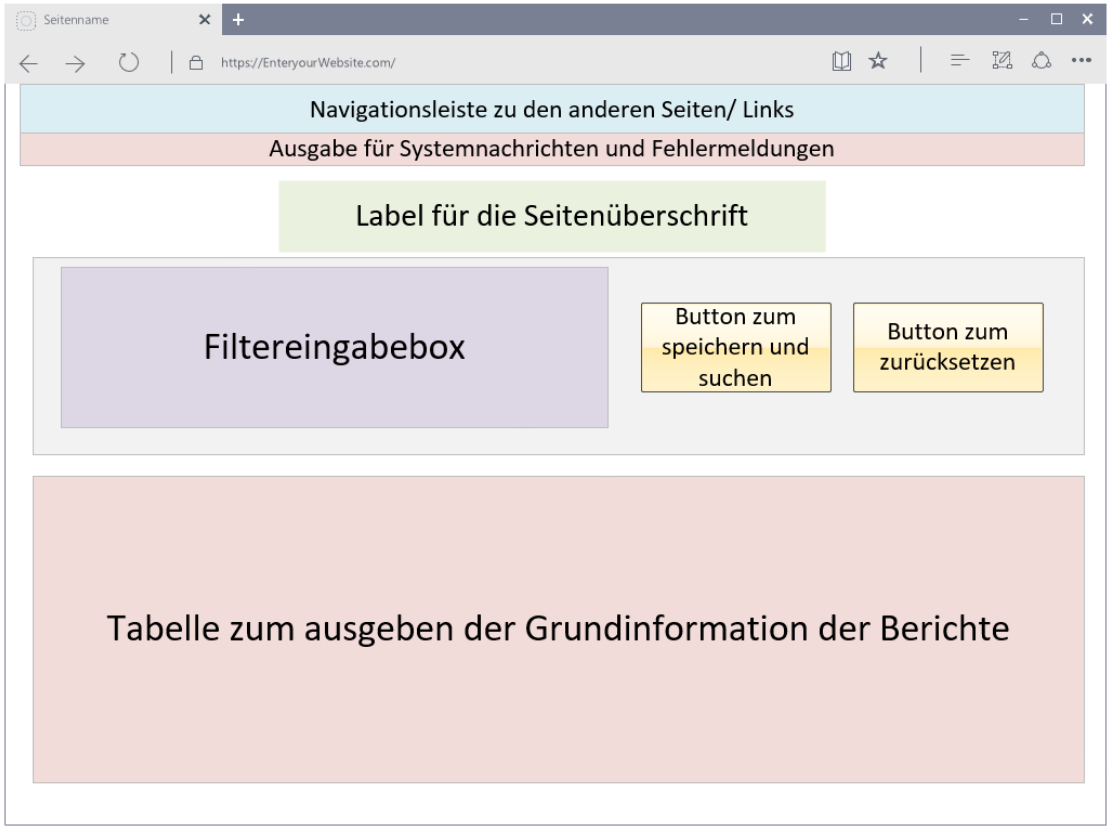


Abbildung 18: Benutzeroberflächenentwurf der Seite zum ausgeben der Berichtstabelle
Quelle: Eigene Darstellung mit Microsoft Visio

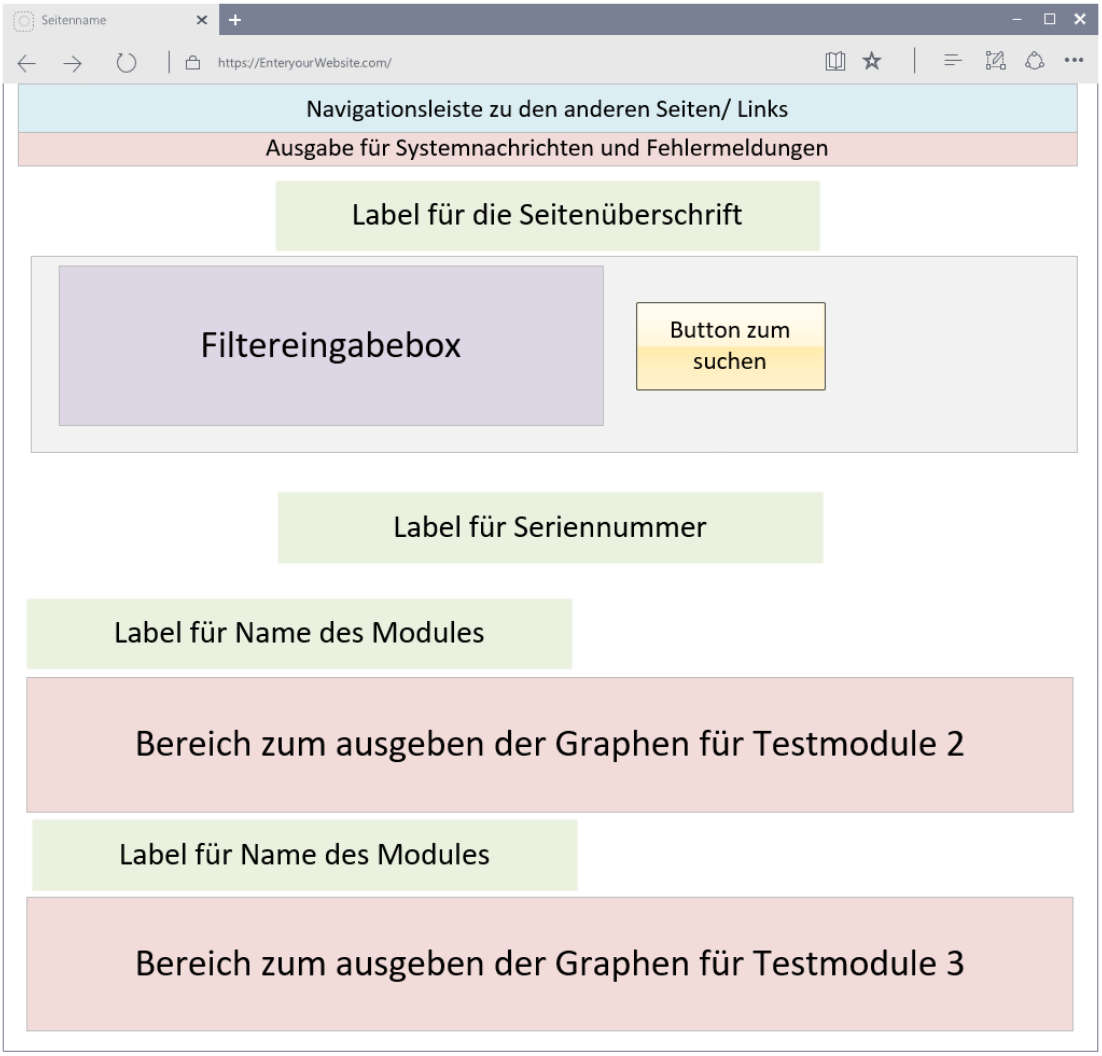


Abbildung 19: Benutzeroberflächenentwurf der Seite zum ausgeben der Graphen
Quelle: Eigene Darstellung mit Microsoft Visio

## 4.4 Entwurf der Applikationsarchitektur

# 5 Implementierung

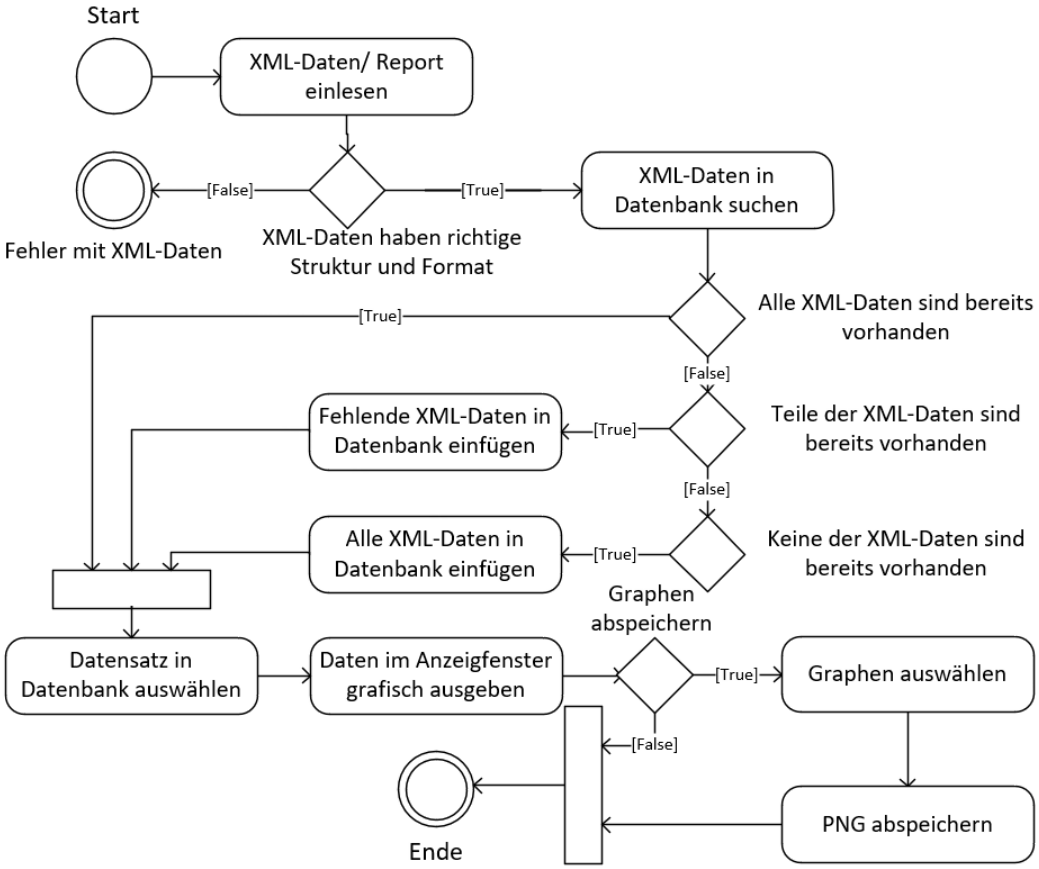


Abbildung 20: Grundlegedes Ablaufdiagramm der Nutzung der Web-Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

## 5.1 Einlesen und Verarbeiten von XML-Daten

## 5.2 Implementierung der Datenbank

## 5.3 Entwicklung der Benutzeroberfläche

## 5.4 Technische Details zur Visualisierung

# 6 Integration und Test

## 6.1 Einbindung in die bestehende Systemlandschaft

## 6.2 Testmethoden und Durchführung

## 6.3 Ergebnisse der Testmethoden

# 7 Fazit und Ausblick

## 7.1 Zusammenfassung der Ergebnisse

## 7.2 Kritische Bewertung

## 7.3 Möglichkeiten für zukünftige Erweiterungen

# i Literaturverzeichnis

# ii Anhangsverzeichnis

# iii Erklärung

Hiermit erkläre ich, dass ich die von mir eingereichte Bachelor- / Masterarbeit "......(Titel der Arbeit).....ßelbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.
Ort und Datum

persönliche Unterschrift

(Name des Verfassers)