

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	IV
Tabellenverzeichnis	V
Abkürzungsverzeichnis	VI
1. Einleitung	1
2. Definition der Anforderungen und ihre Interpretation	3
2.1. Funktionale Anforderungen	3
2.2. Nicht-funktionale Anforderungen	4
2.3. Anforderungen an die Datenbank	5
2.4. Zusammenfassung	6
3. Grundlagen	7
3.1. Überblick über den Umrichter-Prüfstand	7
3.1.1. Aufbau des Teststandes	7
3.1.2. Testmodule	8
3.1.3. Testablauf	9
3.2. Verarbeitung von XML-Daten	11
3.2.1. XML-Strukturaufbau	11
3.2.2. Verarbeiten von XML-Dateien	12
3.2.2.1. XML-Paser	13
3.2.2.2. Adressierung mit XPath	14
3.2.2.3. Pythonbibliotheken für XML-Verarbeitung	16
3.3. Datenbankentwurf und Normalisierung	18
3.3.1. Datenbankstruktur	18
3.3.2. Datenintegrität und Normalisierung	19
3.3.3. Vorgehen zur Erstellung der Datenbank	19
3.4. Grundlagen der Datenvisualisierung	21
3.4.1. Begriff und Zielsetzung	21
3.4.2. Theoretische Grundlagen	21
3.4.3. Formen der Datenvisualisierung	21
3.4.4. Qualitätskriterien	22
3.4.5. Technologische Aspekte	22
3.4.6. Graphen erstellung mit JS	22
3.5. Anforderungen an modulare Softwareentwicklung	24
3.5.1. Klare Modulabgrenzung und Verantwortlichkeiten	24
3.5.2. Geringe Kopplung und hohe Kohäsion	24
3.5.3. Einheitliche Schnittstellen	24
3.5.4. Wiederverwendbarkeit	24
3.5.5. Erweiterbarkeit und Anpassungsfähigkeit	25
3.5.6. Testbarkeit	25
3.5.7. Konsistenz und Standardisierung	25
4. Analyse und Konzeption	26
4.1. Analyse der generierten XML-Berichte und bestehenden Strukturen	26
4.1.1. XML-Berichtsstrukturschema	26
4.2. Datenbankdesign und Strukturkonzeption	30
4.3. Grundkonzept des Benutzeroberflächen-Design	32

4.4. Entwurf der Applikationsarchitektur	35
4.4.1. Architekturübersicht	35
4.4.2. Kommunikation und Datenfluss	39
5. Implementierung	40
5.1. Einlesen und Verarbeiten von XML-Daten	40
5.2. Implementierung der Datenbank	40
5.3. Entwicklung der Benutzeroberfläche	40
5.4. Technische Details zur Visualisierung	40
6. Integration und Test	41
6.1. Einbindung in die bestehende Systemlandschaft	41
6.2. Testmethoden und Durchführung	41
6.3. Ergebnisse der Testmethoden	41
7. Fazit und Ausblick	42
7.1. Zusammenfassung der Ergebnisse	42
7.2. Kritische Bewertung	42
7.3. Möglichkeiten für zukünftige Erweiterungen	42
i. Literaturverzeichnis	i
ii. Anhangsverzeichnis	iii
iii. Erklärung	iv

Abbildungsverzeichnis

1.	Aufbau des Teststandes	8
2.	XML Prolog Beispielcode	11
3.	XML Elemente Beispielcode	12
4.	XML Attribute Beispielcode	12
5.	Beispiel XPath-Ausdruck	16
6.	Aufbau unveränderlicher XML-Berichtstruktur	26
7.	Elementaufbau aus XML-Bericht	27
8.	Direkte Kinderelement unter Stammelement	27
9.	XML-Strukturbeispiel Info-Element	28
10.	XML-Strukturbeispiel Testbench-Element	28
11.	XML-Strukturbeispiel Testmodulheader	29
12.	Beispiel XPowerTest Floatblock-Elemente	29
13.	Beispiel Fehler bei DriverConsumptionTest	30
14.	ER-Modell Überlegung der Datenbankstruktur	31
15.	Darstellung der Datenbanktabellen und ihrer Verbindungen	32
16.	Benutzeroberflächenentwurf der Seite zum Einlesen der XML-Struktur	33
17.	Benutzeroberflächenentwurf der Seite zum Ausgeben der Berichtstabelle	34
18.	Benutzeroberflächenentwurf der Seite zum Ausgeben der Graphen	35
19.	schematisches Architekturdiagramm der Applikation	36
20.	Verzeichnisstruktur der Flask-Applikation	37
21.	Grundlegendes Ablaufdiagramm der Nutzung der Web-Applikation	40

Tabellenverzeichnis

1. Übersicht über XPath-Achsen	15
--	----

Abkürzungsverzeichnis

DUTs	Devices-Under-Test
XML	Extensible Markup Language
DTD	Document Type Definition
API	Application Programming Interface
DOM	Document Object Model
SAX	Simple API for XML
FSW	Functional-Switching
OCP	Over-Current-Protection
DSCP	Dynamic-Short-Circuit-Protection
BIT	Burn-In-Test
OTP	Overtemperature-Protection-Test
XSLT	Extensible Stylesheet Language Transformations
XSD	XML Schema Definition
XXE	XML External Entity
SQL	Structured Query Language

1. Einleitung

„Data is the new oil. Like oil, data is valuable, but if unrefined it cannot really be used.“ Clive Humby (2006)

Die Aufbereitung von Daten, sowohl in ihrer Struktur als auch in ihrer Darstellung, ist entscheidend, um technische Prozesse zu verstehen und effizient mit großen Datenmengen arbeiten zu können. Das Erfassen, Aufbereiten und Interpretieren von Messdaten ist heute aus der Industrie und Wissenschaft nicht mehr wegzudenken. In nahezu allen technischen Bereichen werden Daten erhoben, sei es in der Forschung, bei Qualitätsprüfungen oder in industriellen Testverfahren. Damit diese Daten nutzbar sind, müssen sie verständlich aufbereitet und übersichtlich dargestellt werden.

Die Verarbeitung und Visualisierung großer Datenmengen erfolgt heutzutage meist mithilfe spezialisierter Softwarelösungen. Viele dieser Systeme sind jedoch allgemein gehalten und nur begrenzt auf spezifische technische Anwendungen anpassbar. Das kann sich negativ auf Effizienz, Benutzerfreundlichkeit und Aussagekraft der Ergebnisse auswirken. Besonders in Bereichen, in denen die Datensätze komplex und individuell strukturiert sind, wie bei der Prüfung von Leistungselektronik aus Windenergieanlagen, besteht Bedarf an speziell angepassten Softwarelösungen.

Ziel dieser Arbeit ist es, eine Web-Applikation zu entwickeln, die automatisch erzeugte XML-Berichte eines Umrichter-Teststandes einliest, die enthaltenen Mess- und Gerätedaten in einer Datenbank speichert und diese anschließend grafisch aufbereitet. Auf diese Weise sollen die Daten besser ausgewertet und sowohl für interne Analysen als auch für externe Berichte genutzt werden können. Dabei liegt der Fokus auf einer übersichtlichen Benutzeroberfläche, einer modularen Erweiterbarkeit und einer einfachen Integration in die bestehende Systemlandschaft des Unternehmens.

Die Arbeit entstand in Zusammenarbeit mit einem Serviceanbieter für die Instandhaltung von Windenergieanlagen. Der zugrunde liegende Teststand wird zur Überprüfung von Umrichtern eingesetzt, die aus bestehenden Anlagen stammen. Mithilfe spezieller Prüfverfahren wird bestimmt, ob diese Geräte nach der Instandsetzung wiederverwendet werden können. Die dabei entstehenden XML-Dateien enthalten sämtliche Messwerte, Parameter und Prüfstandsinformationen und bilden die Grundlage für die zu entwickelnde Software.

Für die Umsetzung wurde ein iteratives Vorgehen gewählt. Das bedeutet, dass die Anwendung schrittweise entwickelt und nach jedem Zwischenschritt getestet und verbessert wird. Dieses Vorgehen orientiert sich an Prinzipien agiler Softwareentwicklung, um flexibel auf mögliche Anpassungen reagieren zu können. So entsteht eine funktionierende Anwendung, die sich im Laufe der Entwicklung immer weiter verfeinern lässt.

Die Arbeit gliedert sich in sieben Kapitel. Nach dieser Einleitung werden in Kapitel 2 die funktionalen und nicht-funktionalen Anforderungen beschrieben. Kapitel 3 behandelt die theoretischen und technologischen Grundlagen, die für die Entwicklung der Anwendung relevant sind. In Kapitel 4 folgen die Analyse der vorhandenen Strukturen und der Entwurf der Systemarchitektur. Kapitel

5 beschreibt die Implementierung der Applikation, während Kapitel 6 die Integration in die bestehende Systemumgebung und die Testdurchführung erläutert. Kapitel 7 fasst die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Erweiterungen.

2. Definition der Anforderungen und ihre Interpretation

In diesem Kapitel werden die funktionalen und nicht-funktionalen Anforderungen der zu entwickelnden Web-Applikation beschrieben. Sie wurden in Abstimmung mit den späteren Nutzerinnen und Nutzern sowie den betreuenden Personen im Unternehmen festgelegt. Die Anforderungen bilden die Grundlage für den Entwurf und die Umsetzung der Software und dienen dazu, deren Zielsetzung, Funktionsumfang und technische Rahmenbedingungen klar zu definieren.

2.1. Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die vorgesehenen Funktionen und Abläufe der Applikation. Sie definieren, was die Anwendung leisten soll und wie die Nutzerinnen und Nutzer mit ihr interagieren können.

1. Navigation und Benutzerführung

Die Applikation soll eine einfache und übersichtliche Navigation ermöglichen. Der Wechsel zwischen den Hauptfunktionen erfolgt über eine feststehende Menüleiste im oberen Bereich der Benutzeroberfläche. Dadurch können die Bereiche *Upload*, *Berichte* und *Diagramme* direkt aufgerufen werden.

2. Einlesen von XML-Dateien

Das Einlesen der automatisch generierten XML-Berichte erfolgt über eine Upload-Seite. Die Anwendung soll sowohl ältere als auch neuere XML-Strukturen verarbeiten können, da der Teststand unterschiedliche Versionen erzeugt.

3. Datenverarbeitung und Speicherung

Nach dem Upload sollen die XML-Dateien automatisch geparkt, validiert und in die Datenbank überführt werden. Doppelte Einträge sollen erkannt und vermieden werden.

4. Anzeige der Berichte

Die gespeicherten Berichte sollen tabellarisch dargestellt werden. Jede Zeile repräsentiert einen Testbericht und zeigt die wichtigsten Informationen wie Datum, Seriennummer, Materialnummer und Testergebnis an.

5. Filter- und Suchfunktionen

Die Berichtstabelle soll über Filteroptionen verfügen, um gezielt nach bestimmten Kriterien (z. B. Datum, Seriennummer, Testmodul oder Ergebnisstatus) zu suchen.

6. Grafische Darstellung der Testdaten

Die Anwendung soll die Messergebnisse der einzelnen Module grafisch darstellen. Die Diagramme sind nach Modulen sortiert und enthalten eine klare Achsenbeschriftung sowie Legenden. Das Layout der Diagramme orientiert sich an den Graphen des Teststandprogramms, um den Nutzerinnen und Nutzern die Interpretation zu erleichtern.

7. Systemmeldungen und Fehlermanagement

Nach erfolgreichen oder fehlerhaften Aktionen (z. B. Upload, Datenbankeintrag, Filterabfrage) sollen entsprechende Systemmeldungen angezeigt werden, um den Benutzerstatus transparent zu machen.

8. Exportfunktionen (NEU)

Die Anwendung soll die Möglichkeit bieten, ausgewählte Datensätze oder Berichte in ein externes Format zu exportieren (z. B. CSV oder Excel). Dadurch können die Daten auch außerhalb der Anwendung weiterverarbeitet werden.

9. Modulare Erweiterbarkeit (NEU)

Neue Funktionen, etwa zusätzliche Testmodule oder Analysefunktionen, sollen über separate Module oder Blueprints ergänzt werden können, ohne bestehende Teile der Anwendung zu verändern.

10. Benutzerverwaltung (optional) (NEU)

Die Architektur der Applikation soll es ermöglichen, zukünftig eine einfache Benutzerverwaltung zu integrieren. Damit können verschiedene Rollen (z. B. Administrator, Nutzer, Gast) eingeführt werden.

2.2. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Qualitätsmerkmale der Software. Sie legen fest, unter welchen Bedingungen die Applikation betrieben werden soll und welche Eigenschaften sie erfüllen muss.

1. Plattform und Laufzeitumgebung

Die Web-Applikation wird auf einem unternehmenseigenen Apache2-Server betrieben. Sie basiert auf Python und dem Webframework Flask.

2. Performance

Das System muss auch bei größeren XML-Dateien zuverlässig arbeiten. Eine aktive Laufzeitoptimierung ist nicht erforderlich, solange der Arbeitsfluss nicht beeinträchtigt wird.

3. Usability

Die Benutzeroberfläche soll intuitiv bedienbar sein und eine klare Struktur aufweisen. Eingabefehler sollen vermieden und, wenn möglich, automatisch erkannt werden.

4. Modularität und Wartbarkeit

Die Applikation ist modular aufgebaut, um eine einfache Pflege und spätere Erweiterung zu ermöglichen. Änderungen an einzelnen Komponenten sollen keine Anpassungen an anderen Modulen erfordern.

5. Sicherheit und Datenintegrität (NEU)

XML-Dateien müssen vor dem Einlesen auf korrekte Struktur und mögliche Sicherheitsrisiken geprüft werden (z. B. Schutz vor fehlerhaften oder manipulierten Dateien).

6. Fehler- und Ausnahmebehandlung (NEU)

Unerwartete Systemfehler sollen abgefangen und im Log gespeichert werden. Für Benutzerinnen und Benutzer werden stattdessen verständliche Fehlermeldungen ausgegeben.

7. Skalierbarkeit (NEU)

Das System soll bei Bedarf mit minimalem Aufwand um zusätzliche Funktionen, Testmodule oder Datenquellen erweitert werden können.

8. Kompatibilität (NEU)

Die Applikation soll auf allen gängigen Browsern lauffähig sein (z. B. Chrome, Edge, Firefox).

9. Dokumentation und Nachvollziehbarkeit (NEU)

Der Code soll übersichtlich dokumentiert werden. Wichtige Abläufe (Upload, Datenverarbeitung, Visualisierung) werden nachvollziehbar in der Projektdokumentation beschrieben.

2.3. Anforderungen an die Datenbank

Die Datenbank ist ein zentraler Bestandteil der Anwendung. Sie speichert alle relevanten Testdaten, Parameter und Informationen aus den XML-Berichten.

1. Die Datenbank basiert auf dem relationalen Datenbanksystem **MariaDB**.
2. Sie muss in der Lage sein, sämtliche in den XML-Berichten enthaltenen Daten vollständig abzubilden, um die Wiederherstellung eines gesamten Testberichts theoretisch zu ermöglichen.
3. Datendopplungen sollen vermieden werden. Häufig vorkommende Informationen (z. B. Teststandname, Versionsnummern, DUT-Typen) werden ausgelagert und über Fremdschlüssel referenziert.
4. Für DUT-Typen und Seriennummern werden separate Informationstabellen angelegt, um Redundanzen zu vermeiden und die Datenstruktur klar zu halten.
5. Die Datenbankstruktur soll so gestaltet sein, dass zukünftige Teststände oder zusätzliche Testmodule problemlos integriert werden können.
6. **Verwendung von ORM-Technologie (NEU)**
Für die Kommunikation zwischen Anwendung und Datenbank wird **SQLAlchemy** eingesetzt. Dadurch kann der Datenzugriff objektorientiert erfolgen, was die Wartbarkeit und Lesbarkeit des Codes verbessert.
7. **Versionierung und Migration (NEU)**
Änderungen an der Datenbankstruktur werden mit **Alembic** verwaltet, um eine konsistente Weiterentwicklung der Datenbank sicherzustellen.

8. Datenvalidierung und Konsistenzprüfung (NEU)

Beim Einfügen neuer Datensätze sollen Validierungen sicherstellen, dass die Werte logisch und formal korrekt sind (z. B. keine Nullwerte bei Pflichtfeldern, korrekte Datentypen).

2.4. Zusammenfassung

Die beschriebenen Anforderungen bilden den Rahmen für die Entwicklung der Web-Applikation. Während die funktionalen Anforderungen die konkreten Aufgaben und Abläufe definieren, legen die nicht-funktionalen Anforderungen fest, wie die Anwendung qualitativ umgesetzt werden soll. Zusammen mit den Datenbankanforderungen bilden sie die Grundlage für den folgenden Entwurf der Systemarchitektur und die spätere Implementierung.

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. [1]

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 Devices-Under-Test (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 1 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 1 mit ER bezeichnet, für „Electronic Rack“.
- Der Testmatrix-Schrank, in dem die Sammelschienen für den Stromanschluss und die Schützen sitzen. In Abbildung 1 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 1 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 1 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. [1]

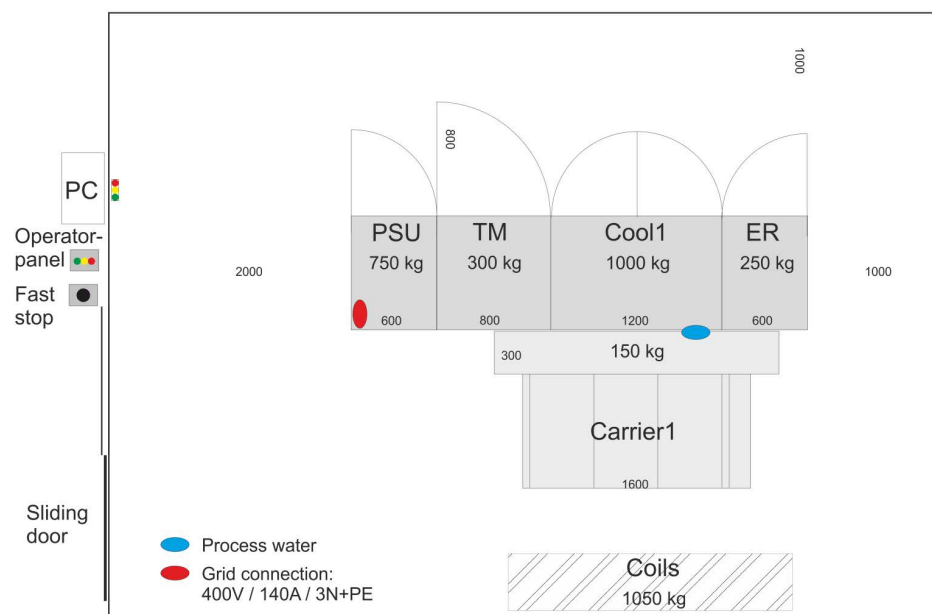


Abbildung 1: Aufbau des Teststandes

Quelle: [1, S. 7]

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 Functional-Switching (FSW) kann überprüft werden, ob die Halbleiter, die sich in den DUTs befinden, generell schalten.

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

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

- Burn-In-Test (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 Overtemperature-Protection-Test (OTP) wird ein DUT, ähnlich wie beim BIT, nur mit reduzierter Kühlung betrieben, bis die maximal zulässige Kühlkörpertemperatur erreicht ist und die Temperaturschutzschaltung auslöst.[1]

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. [1]

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 Extensible Markup Language (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 die Arbeit relevante Aspekte von dem Dokumententyp XML, da die Messwerte bzw. die Berichte die der Teststand generiert in 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 [2]. 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.[3, S. 4]

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 war das angestrebte Ziel Daten in einem einheitlichen Standard zwischen verschiedenen Anwendungen und Plattformen zu speichern und auszutauschen zu können.[3, S. 3–5]

3.2.1. XML-Strukturaufbau

Eine XML-Datei beginnt mit Prolog, der die XML-Version und die verwendete Zeichencodierung definiert. In Abbildung 2 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 `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 Document Type Definition (DTD) deklariert werden, dies sind jedoch optional und für diese Arbeit nicht weiter relevant. [4, S. 8, 9]

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

Abbildung 2: 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-Dateien einer hierarchischen Baumstruktur folgen. Hierbei wird das oberste Element als Wurzelement bezeichnet, im Englischen „root element“.[4, S. 10–14]

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

Abbildung 3: 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 4 gezeigt. [4, S. 10–14]

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

Abbildung 4: 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 [4, S. 10–14]. Dies ist für die automatisch generierten Berichte irrelevant, jedoch für das Beschreiben der Beispiele hilfreich. In Abbildung 3 und Abbildung 4 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.[4, S. 10]

Paser werden verwendet um eine Applikation über eine Application Programming Interface (API) eine Schnittstelle auf ein XML-Dokument zu geben. Bei APIs wird in diesem Bereich zwischen zwei Grundtypen unterschieden:[4, S. 405]

Die baumbasierten APIs 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 APIs ist Document Object Model (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.[4, S. 413, 414]

Die ereignisbasierten APIs 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 treten 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 APIs besonders gut für die Verarbeitung großer XML-Dateien, da dies für eine geringe Arbeitsspeicherbelastung sorgt. Trotz der Effizienz hinsichtlich des Speicherverbrauchs sind ereignisbasierte APIs für die Datenmanipulation nicht sonderlich gut geeignet. Es wird nämlich durch das Teilweise einlesen keine umfassende Dokumentstruktur wie beim aufgebaut Die baumbasierten APIs, welches die Manipulation bzw. bearbeitung erschwert. Ein Beispiel für eine große ereignisbasierte API ist Simple API for XML (SAX). Diese API abeitet 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.[4, S. 405]

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 bestimmte Knoten zu adressieren. Aus diesem Grund wurde die Abfragesprache XML Path Language kurz XPath entwickelt. XPath wird hauptsächlich in der Transformationssprache Extensible Stylesheet Language Transformations (XSLT) eingesetzt, welche XML-Dateien in andere 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 verwendet. Knoten beschreiben Teile des XML-Dokumentes können, aber auch über das Wurzelknoten das gesamte Dokument beschreiben. Knoten können alle im XML-Dokument vorhandenen strukturellen 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 diese Knoten kann man durch die XML-Struktur navigieren, aber um nach den Knoten zu adressieren werden Achsen benötigt. Achsen spezifiziert die Beziehung zwischen den Knoten, um so die gewünschten Knoten zu selektieren. Es gibt Achsen, welche nur einen Knoten adressieren, so wie Achsen, welche mehrere Knoten gleichzeitig auswählen. In der folgenden Tabelle 1 werden die verschiedenen Achsen benannt und beschrieben. Für alle Achsen, die keine Kurz-Notation enthalten, muss im Code der Name ganz ausgeschrieben werden. [5]

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 [5]

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 diesen Operatoren und Aneinanderreihungen von diesen kann eine Vielzahl von Sucheinschränkungen geschaffen werden, welche die Genauigkeit der Suchen stark erhöhen und variable gestalten. [5]

Ein XPath-Ausdruck, also ein Path welcher zur Adressierung von einem oder mehreren Knoten verwendet wird, besteht aus einem oder mehreren Lokalisierungsschritten. Ein Lokalisierungsschritt besteht aus weiteren 3 Teilen: der Achse, einem Knotentest und bei Bedarf aus einem oder mehreren Prädikaten. Der Knotentest ist hierbei nichts anderes als der Name des 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 Knotentests bzw. des Knotennamens das Wildcard-Zeichen * in einem Lokalisierungsschritt verwendet wird, werden unabhängig vom Namen und Achsen alle Knoten selektiert, welche sich unter dem letzten Lokalisierungsschritt befinden.

Die Lokalisierungsschritte werden mithilfe des Schrägstrichs / voneinander getrennt. [5] In der folgenden Abbildung 5 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 5: 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 bieten weniger Funktionen. Im folgen wird einige wichtige Bibliotheken genannt und kurz erläutert:

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.[6]

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. [7]

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.[8]

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.[9]

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. [10]

xmlschema

Spezialisiert auf XML Schema Definition (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. [11]

defusedxml

Eine sichere Variante der Standardbibliotheken. Schützt vor bekannten XML-Sicherheitslücken wie „Billion Laughs“ oder XML External Entity (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. [12]

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. [13]

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 ermöglichen, sowie redundante Datenhaltung vermeiden und die effiziente Abfragen über spezielle Sprachen wie die Structured Query Language (SQL) ermöglichen.[14, S. 6]

In der folgen werden die Hauptanforderungen an Datenbanken bzw. Datenbankmanagementsysteme kurz zusammengefasst: [15, S. 7],[14, S. 6]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: [16, S. 35]

- 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: [16, S. 38–41]

- 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: [16, S. 38–41]

- 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: [15, S. 9–11]

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 bestehenden 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 Zellenkonfigurationen (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 6 ist die unveränderliche Struktur eines Testberichtes abgebildet, welcher erfolgreich durchgeführt wurde.

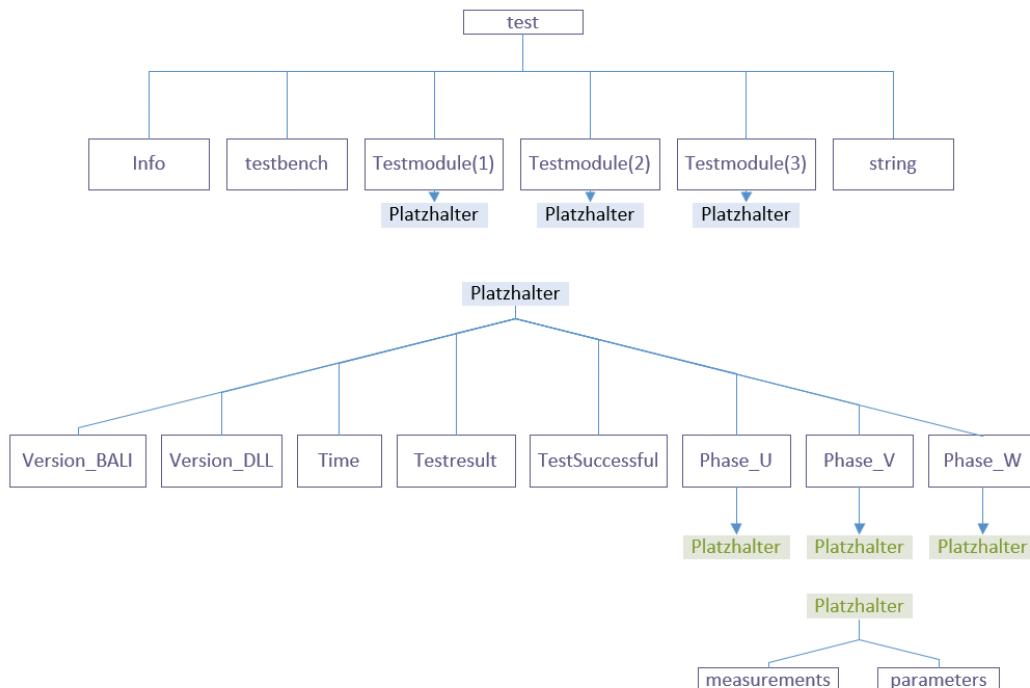


Abbildung 6: Aufbau unveränderlicher XML-Berichtstruktur

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 7.

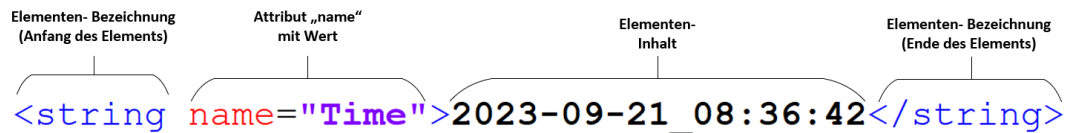


Abbildung 7: Elementaufbau aus XML-Bericht

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 8 zu erkennen.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2   <test id="124">
3     <info></info>
4     <testbench></testbench>
5     <testmodule name="DriverConsumptionTest"></testmodule>
6     <testmodule name="PulseTest_FSW_L"></testmodule>
7     <testmodule name="XPowerTest"></testmodule>
8     <string name="test_sequence_status">
9       Test sequence successfully finished.</string>
10  </test>

```

Abbildung 8: 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 9

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

Abbildung 9: 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 10 zu sehen.

```
1 <testbench>
2     <string name="Testbench_name">USTB_DtWind</string>
3     <string name="Testbench_HW_revision">V1.5</string>
4     <string name="Version_GUI">V1.4.1</string>
5     <string name="Version_controller">V1.3.2</string>
6 </testbench>
```

Abbildung 10: 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 Abbildung 11.

```

1 <testmodule name="DriverConsumptionTest">
2     <integer name="TestSuccessful" min="1" max="1">1</integer>
3     <string name="Testresult">
4         Test finished successfully!</string>
5     <string name="Time">2023-09-21_08:36:42</string>
6     <string name="Version_DLL">V1.2.4</string>
7     <string name="Version_BALI">V1.2.4</string>
8     <Phase_U>...</Phase_U>
9     <Phase_V>...</Phase_V>
10    <Phase_W>...</Phase_W>
11 </testmodule>

```

Abbildung 11: 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 9 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 Wertaufnahme angeben dies ist in Abbildung 12 zu erkennen. Diese sind für die grafische Darstellung besonders relevant.

```

1 <floatblock name="Time" size="698" duration="959.952" unit="s">
2 0.71365,2.1370,3.5323,...</floatblock>

```

Abbildung 12: 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 13 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.

```
1 <testmodule name="DriverConsumptionTest">
2   <integer name="TestSuccessful" min="1" max="1">0</integer>
3   <string name="Testresult">Idrv not within limits on Phase U
4 0x30010019 !
5 0x30010013</string>
6   <string name="Time">2021-03-10_09:10:37</string>
7   <string name="Version_DLL">V1.2.4</string>
8   <string name="Version_BALI">V1.2.4</string>
9   <Phase_U>...</Phase_U>
10  <Phase_V>...</Phase_V>
11   <Phase_W>...</Phase_W>
12 </testmodule>
```

Abbildung 13: 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 14. 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.

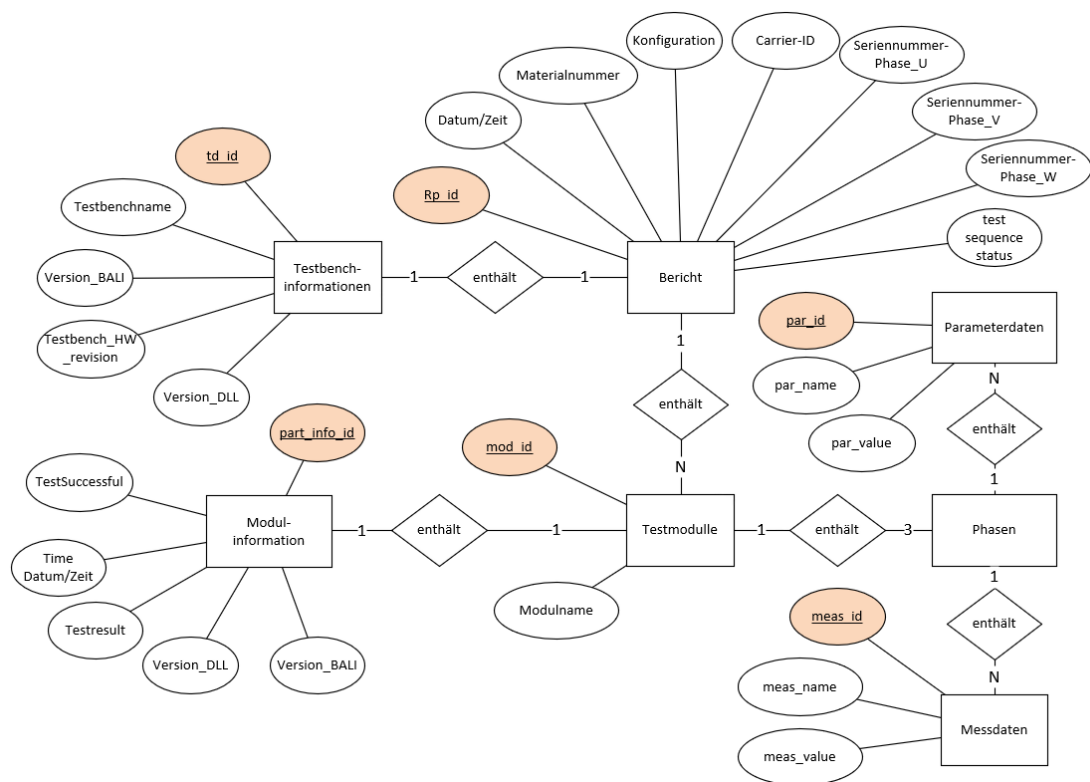


Abbildung 14: ER-Modell Überlegung der Datenbankstruktur

Quelle: Eigene Darstellung mit Microsoft Visio

Aus diesem ER-Modell wurde die Datenbanktabelle und ihre Verbindungen über Fremdschlüssel abgeleitet. In Abbildung 15 werden die Datenbanktabellen, ihre genauen Namensbezeichnungen der Tabelle, dem Datentyp in der Datenbank und ihr Inhalt, sowie ihre Verbindungen abgebildet.

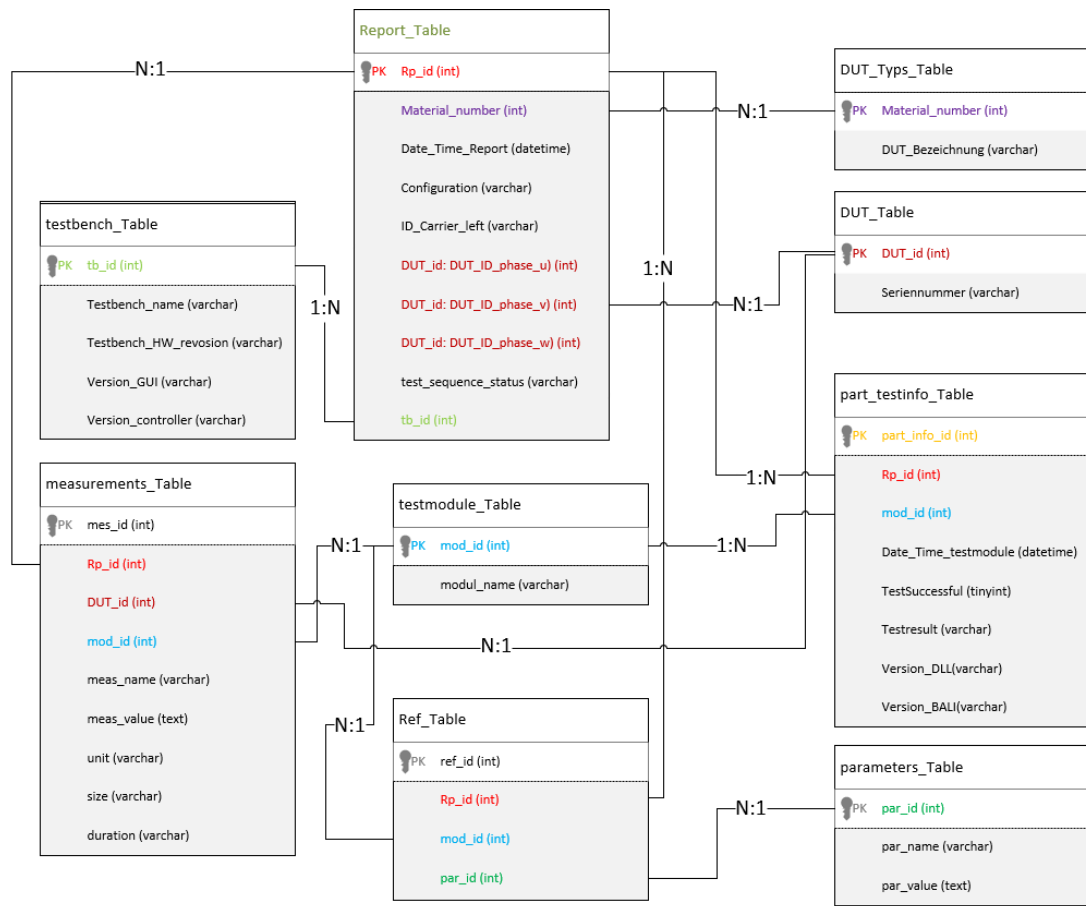


Abbildung 15: Darstellung der Datenbanktabellen und ihrer Verbindungen

Quelle: Eigene Darstellung mit Microsoft Visio

4.3. Grundkonzept des Benutzeroberflächen-Design

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

Die Benutzeroberfläche wird in drei Seiten aufgeteilt, welche sich mit dem Einlesen der Daten, dem darstellen und auswählen zu visualisierenden Daten und dem Ausgeben des Ergebnis befassen. Für das Navigieren zwischen den Seiten soll sich auf jeder Seite eine Navigation-Menü im oberen Bereich der Seite befinden. Zudem sollen alle Seiten einen Bereich für Systemnachrichten wie Bestätigung oder

Fehlermeldungen und ein Label für die Seitenüberschrift besitzen. Diese soll direkt unter den Navigation- Menü angesiedelt werden, um die Grundstruktur identisch aufzubauen. In den nachfolgenden Text werden die Konzepte der Benutzeroberflächen und ein grundlegendes Benutzungskonzept der einzelnen Seiten anhand der Abbildungen 16, 17 und 18 beschrieben.

Für das Einlesen der XML-Daten soll ein Eingabebox genutzt werden, in die die XML-Struktur kopiert wird. Die Eingaben soll über einen Button unter der Eingabebox bestätigt werden. Bei einem erfolgreichen Einlesen und Einfügen in die Datenbank soll eine Bestätigungsnachricht im

Bereich für Systemnachrichten erscheinen. Bei einem Fehler soll in diesem Bereich eine Fehlermeldung mit möglicher Lösung erscheinen. Der Aufbau dieser Seite ist in der Abbildung 16 dargestellt.

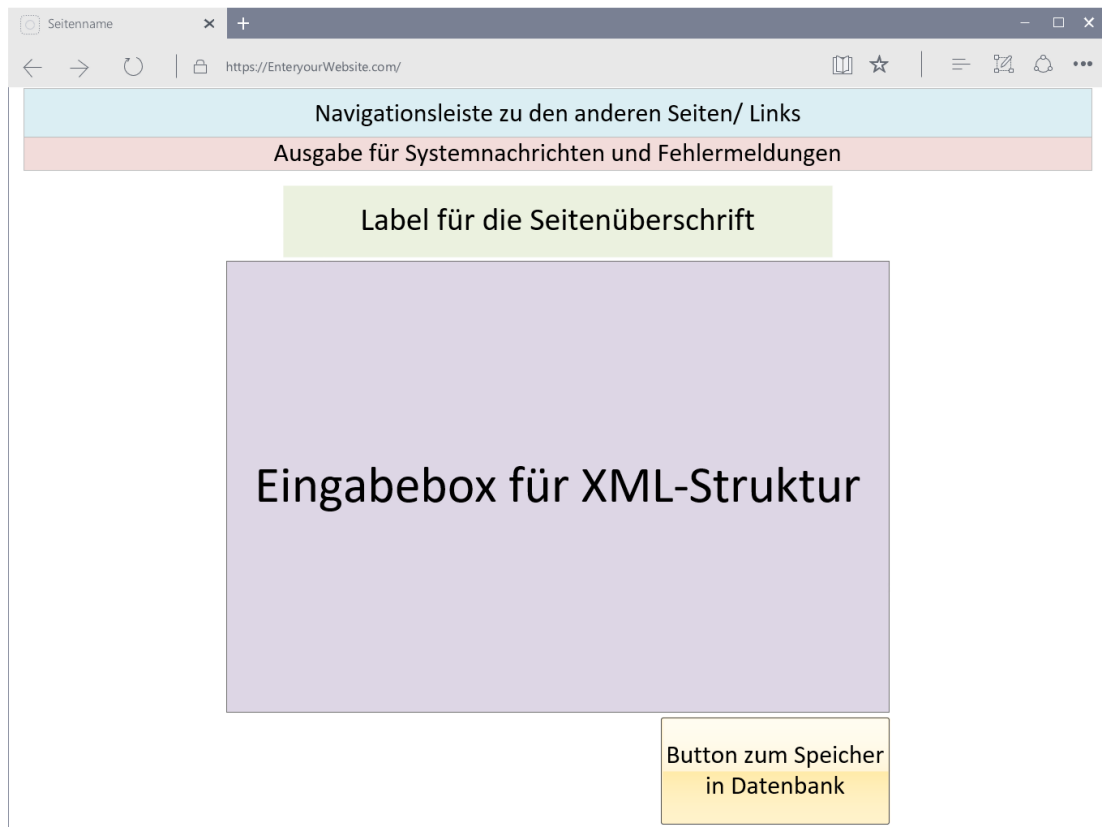


Abbildung 16: Benutzeroberflächenentwurf der Seite zum Einlesen der XML-Struktur

Quelle: Eigene Darstellung mit Microsoft Visio

Die Seite für das Darstellen und Auswählen zu visualisierenden Daten soll im oberen Abschnitt der Seite ein Bereich für Filtereinstellungen besitzen, mit möglichen Filteroption wie Datum oder Seriennummer. Neben dem Filtereinstellung soll zwei Button für das Bestätigen und Zurücksetzen der Filtereinstellungen sein. Bei einem Fehler bei der Filtereingaben soll in Bereich für Systemnachrichten eine Fehlermeldung mit möglicher Lösung erscheinen. Im unteren Bereich der Seite soll sich eine Tabelle, welche die eingelesenen Berichte in der Datenbank anzeigt. Diese soll durch die Filtereinstellungen angepasst werden. Hierbei muss auf eine Anzeige Möglichkeit für länge Tabellenstrukturen berücksichtigt werden, um die Benutzung effizient zu halten. Das Grundkonzept ist in Abbildung 17 dargestellt.

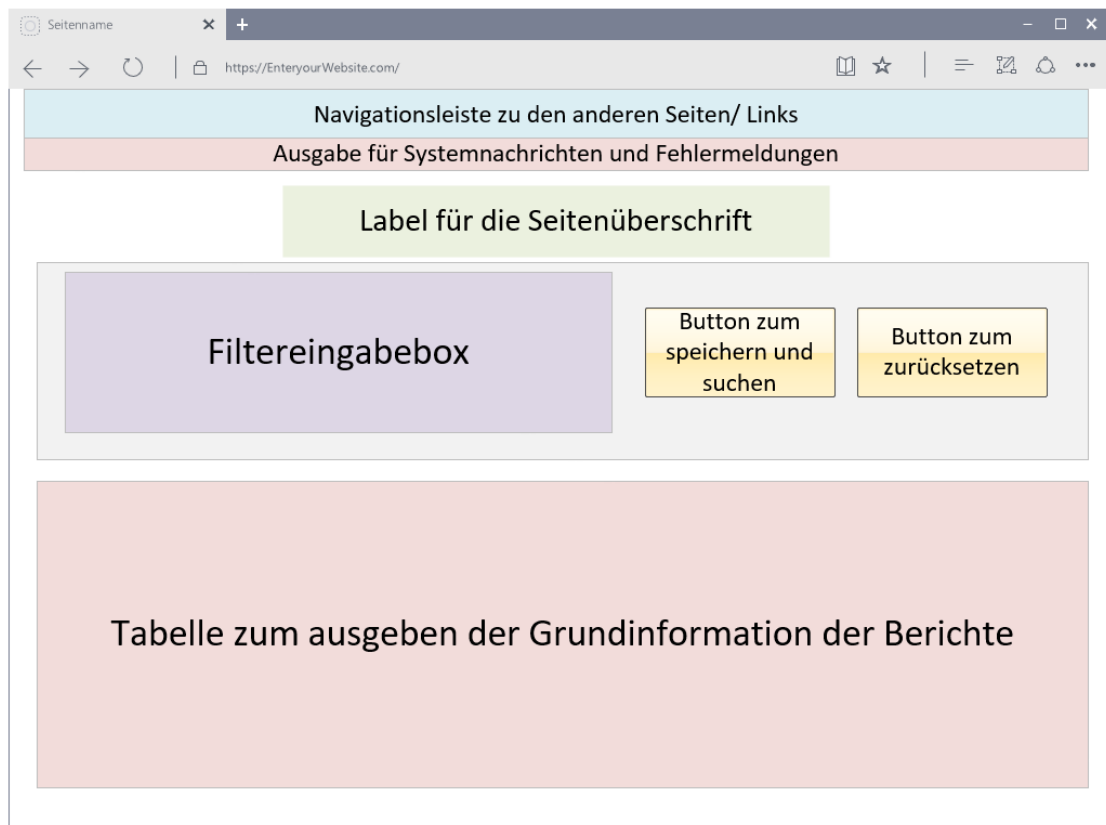


Abbildung 17: Benutzeroberflächenentwurf der Seite zum Ausgeben der Berichtstabelle

Quelle: Eigene Darstellung mit Microsoft Visio

Für das Ausgeben der ausgewählten Berichtsdaten soll im oberen Abschnitt der Seite eine Instanz eingefügt werden, die die Information zu dem Ausgewählten Berichtsdaten angezeigt werden. Im unteren Bereich soll ein Label für die Seriennummer der Ausgewählten Berichtsdaten eingefügt werden. Darunter werden Enthalten den Modulen mit Labeln benannt und unter den Modulname werden die Graphen und Werte für die visuelle Darstellung der Testdaten eingefügt. Die Anzahl der Darstellungen hängt von den in den Bericht Daten enthaltenden Modulen ab. Die Anzahl und der Aufbau des Unteren Bereiches ist variabel. Der grundlegende Aufbau dieser Seite ist in der Abbildung 18 dargestellt.

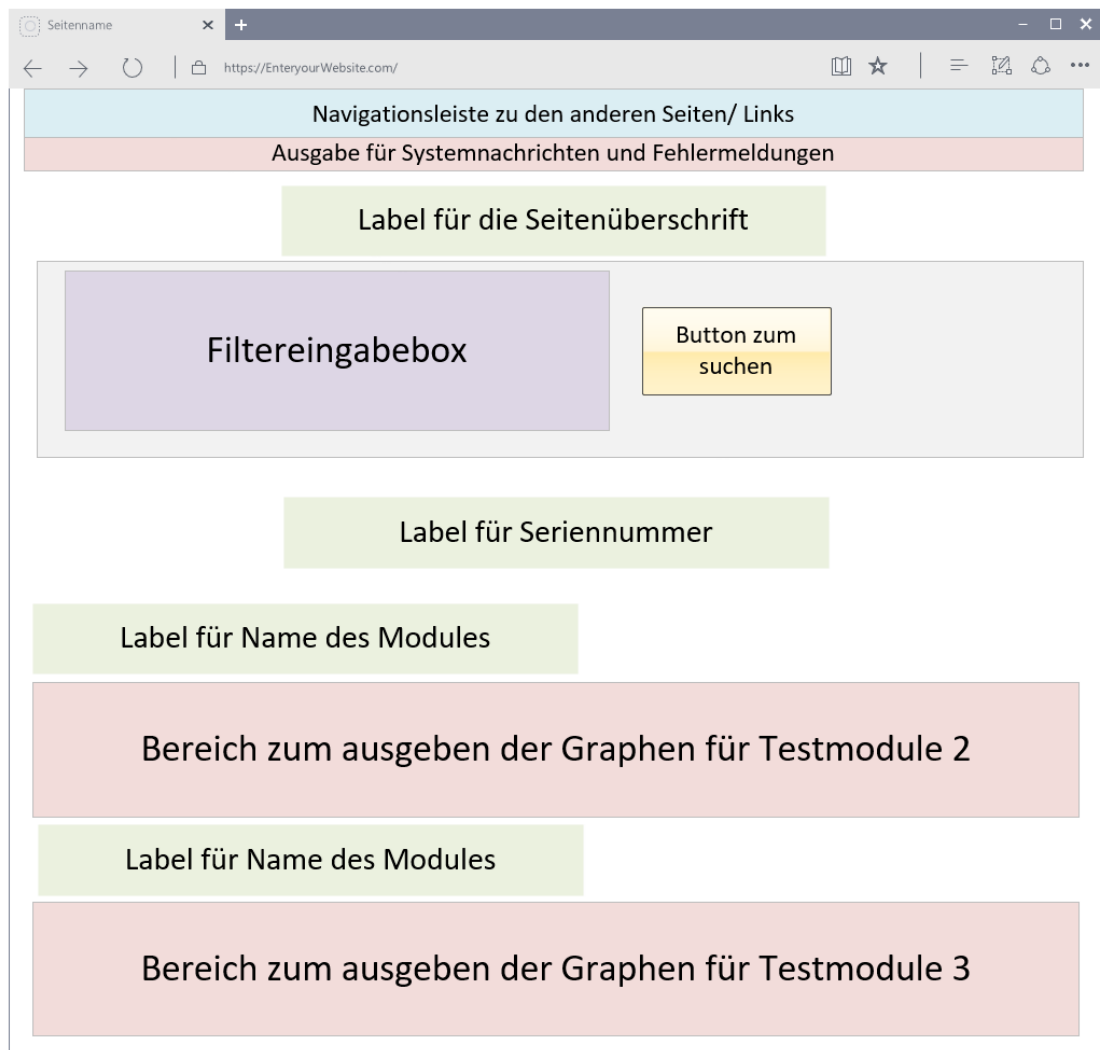


Abbildung 18: Benutzeroberflächenentwurf der Seite zum Ausgeben der Graphen

Quelle: Eigene Darstellung mit Microsoft Visio

4.4. Entwurf der Applikationsarchitektur

Der Entwurf der Applikationsarchitektur bildet die konzeptionelle Grundlage für die technische Umsetzung der entwickelten Web-Applikation. Das Ziel ist es, eine Struktur zu schaffen, die modular, wartbar und erweiterbar ist, und die gleichzeitig den funktionalen Anforderungen gerecht wird und sich in die bestehende Systemlandschaft integrieren lässt. Die Architektur der Anwendung ist schichten- und komponentenorientiert aufgebaut und nutzt das Webframework Flask, welches eine klare Trennung zwischen der Präsentations-, Logik- und Datenhaltungsschicht ermöglicht.

4.4.1. Architekturübersicht

Die Applikation soll als serverbasierte Webanwendung im Client-Server-Modell funktionieren. Das bedeutet: Während der Server die Datenverarbeitung, -speicherung und -bereitstellung über-

nimmt, ermöglicht der Client über den Webbrowser die Darstellung und Interaktion. Die Anwendung ist in mehrere Schichten gegliedert, zu denen man die einzelnen Programmteile zuordnen kann. Die Anwendung wird in folgende Schichten gegliedert:

- Präsentationsschicht (Frontend): Bereitstellung der Benutzeroberfläche über HTML-Templates, CSS und JavaScript.
- Applikationslogik (Backend): Implementierung der Geschäftslogik, Steuerung des Datenflusses und Verarbeitung der XML-Dateien.
- Datenhaltungsschicht: Persistente Speicherung der extrahierten Mess- und Gerätedaten in einer relationalen Datenbank mittels ORM.

Ein schematisches Architekturdiagramm dieser Struktur ist in Abbildung 19 dargestellt.

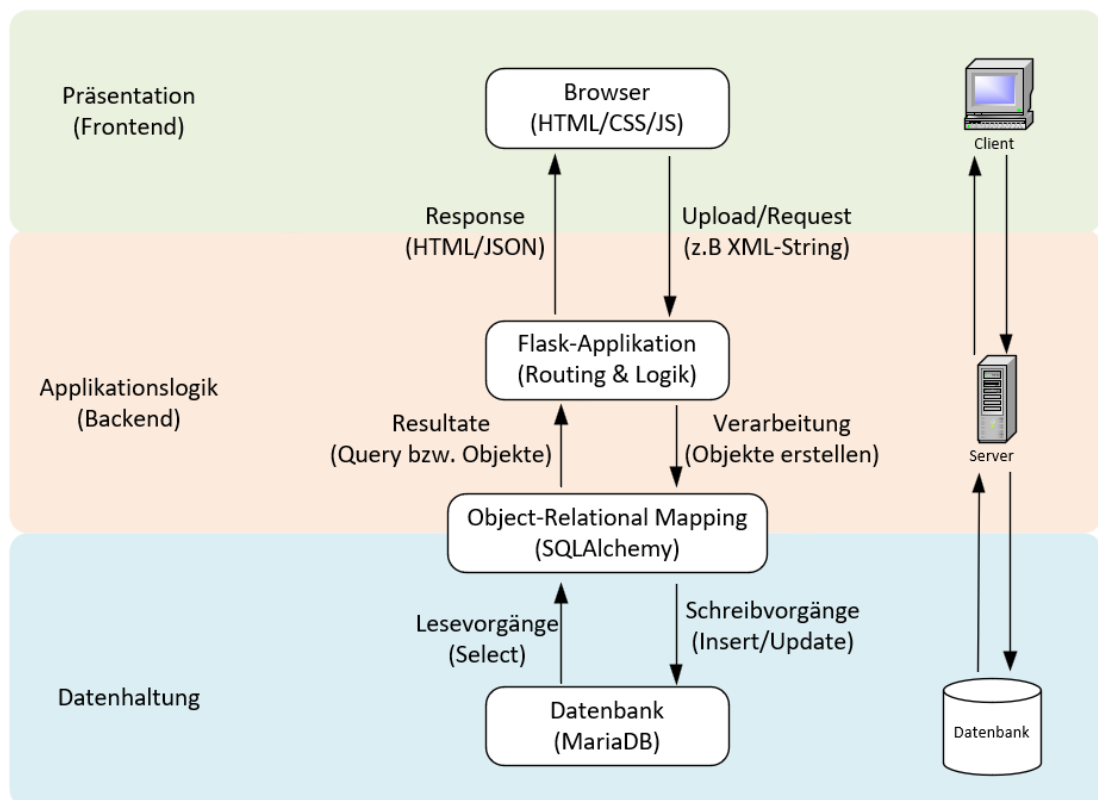


Abbildung 19: schematisches Architekturdiagramm der Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

Präsentationsschicht

Die Abbildung zeigt den Aufbau der Flask-Applikation. Im Verzeichnis `blueprints` befinden sich die einzelnen Routen-Module, während im Ordner `models` die Datenbankmodelle definiert sind.

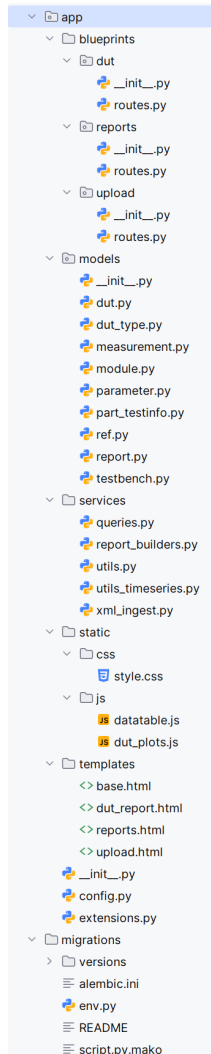


Abbildung 20: Verzeichnisstruktur
der
Flask-
Applikation

Die Präsentationsschicht besteht aus einer Kombination aus HTML-Templates und statischen Ressourcen (CSS, JavaScript), die im Verzeichnis `templates/` bzw. `static/` abgelegt sind.

- Die Templates `upload.html`, `reports.html` und `dut_report.html` bilden die Kernseiten der Weboberfläche.
- Statische Dateien wie `style.css` und `dut_plots.js` dienen der Gestaltung und der interaktiven Darstellung von Messergebnissen.

Die Kommunikation mit der Logikschicht erfolgt über die definierten *Flask Blueprints*, die als modulare Controller agieren.

Applikationslogik

Die zentrale Geschäftslogik ist in modularen *Blueprints* und *Service-Komponenten* realisiert.

Die drei Blueprints `upload`, `reports` und `dut` kapseln die jeweiligen Funktionsbereiche:

- `upload`: Einlesen und Validieren von XML-Dateien.
- `dut`: Verarbeitung und Anzeige der Daten eines „Device Under Test“.
- `reports`: Zusammenstellung und Ausgabe von Auswertungen.

Unterstützt werden diese durch das Modul `services/`, das die eigentliche Logik zur Datenverarbeitung bereitstellt:

- `xml_ingest.py` übernimmt das Parsen und Einlesen der XML-Daten.
- `queries.py` enthält vordefinierte Datenbankabfragen.
- `utils.py` und `utils_timeseries.py` stellen Hilfsfunktionen zur Verfügung, z. B. zur Zeitreihenanalyse und Datenaufbereitung.
- `report_builders.py` dient der Erstellung und Formatierung von Report-Daten für die grafische Darstellung.

Die Applikationslogik wird über die Datei `run.py` initialisiert, welche den Flask-Server startet und die Anwendungskonfiguration aus `config.py` einliest.

Datenhaltungsschicht

Die Datenhaltung erfolgt über ein relationales Datenbanksystem, das über das Flask-eigene *SQLAlchemy ORM* angebunden ist.

Die Datenmodelle sind im Verzeichnis `models/` definiert und bilden die logischen Entitäten der Prüfanlage ab, darunter:

- `dut.py` (Device Under Test)
- `measurement.py` (Messdaten)
- `parameter.py` (Prüfparameter)
- `testbench.py`, `module.py` und `ref.py` (Referenzdaten und Prüfaufbau)

Die Beziehungen zwischen den Modellen ermöglichen eine strukturierte und relationale Abbildung der Prüfdaten, wodurch eine effiziente Abfrage und Analyse möglich ist.

Migrationen werden mit *Alembic* verwaltet, wie das Verzeichnis `migrations/` zeigt.

4.4.2. Kommunikation und Datenfluss

Alles startet mit dem Hochladen einer XML-Datei über die Weboberfläche. Der Blueprint upload übergibt die Datei an das Modul `xml_ingest`, welches die XML-Struktur analysiert, relevante Informationen extrahiert und sie über die Datenmodelle in der Datenbank speichert. Die über den Blueprint reports oder dut gespeicherten Daten können anschließend abgerufen und visuell über die Templates dargestellt werden. Die Schichten kommunizieren ausschließlich über klar definierte Schnittstellen und ORM-Modelle, was eine lose Kopplung und eine hohe Wartbarkeit ermöglicht.

5. Implementierung

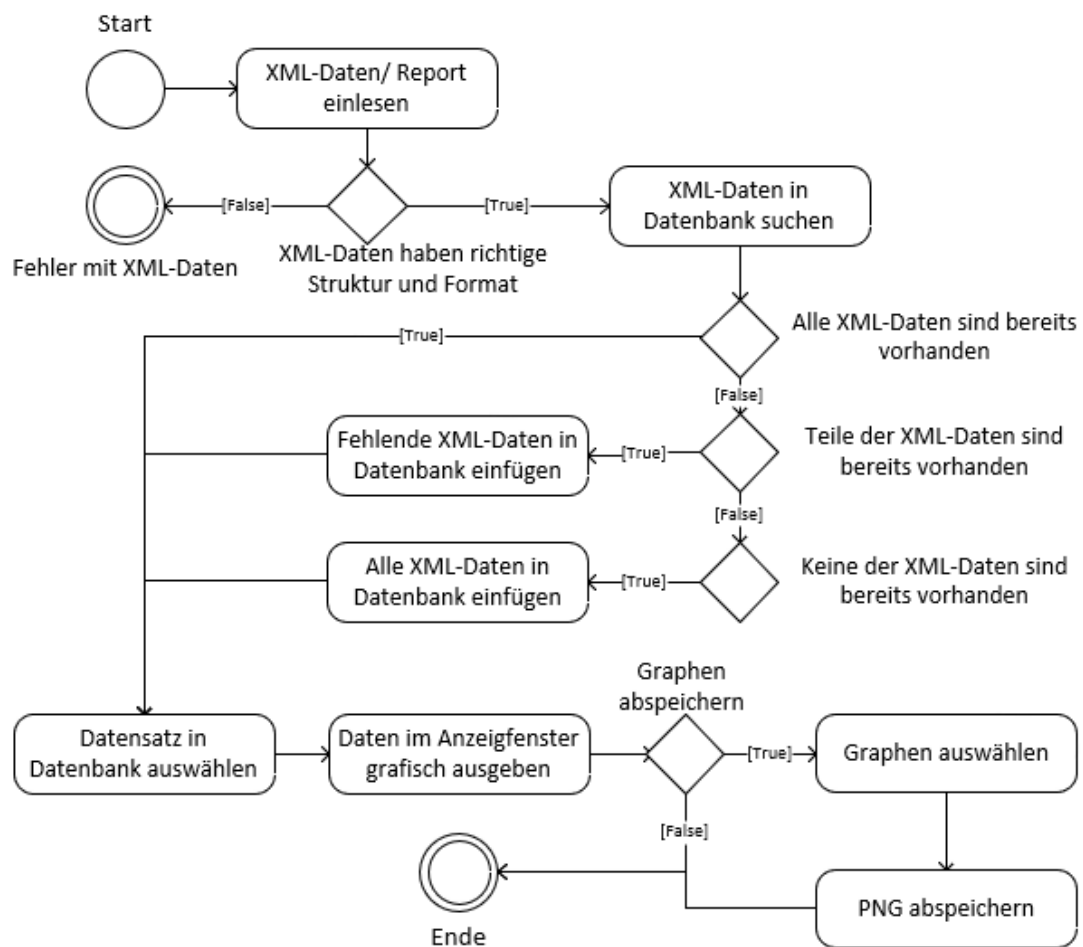


Abbildung 21: Grundlegendes Ablaufdiagramm der Nutzung der Web-Applikation
Quelle: Eigene Darstellung mit Microsoft Visio

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

Literatur

- [1] A. GmbH, *USTB DWT (XCT0006-1) Main Manual V1.0*, Aachen: AixControl GmbH, 2018.
- [2] G. Neumann, „Auszeichnungssprache,“ in *Enzyklopädie der Wirtschaftsinformatik – Online-Lexikon*, Online; abgerufen am 18.08.2025, Berlin: GITO, 2019. besucht am 18. Aug. 2025. Adresse: <https://wi-lex.de/index.php/lexikon/technologische-und-methodische-grundlagen/sprache/auszeichnungssprache/>
- [3] P. Brezany, *XML und Datenbanken*, Vorlesungsskript, Online; abgerufen am 22.09.2025, 2003. besucht am 22. Sep. 2025. Adresse: <https://homepage.univie.ac.at/peter.brezany/teach/kfk/02ws-vo/skriptum/14-01-03.pdf>
- [4] M. Becher, „XML-Grundlagen,“ in *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM*. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, ISBN: 978-3-658-35435-0. DOI: 10.1007/978-3-658-35435-0_1 Adresse: https://doi.org/10.1007/978-3-658-35435-0_1
- [5] B. Jung. „Homepage Webhilfe: Grundlagen XPath.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://www.homepage-webhilfe.de/XML/XPath/>
- [6] Python Software Foundation. „xml.etree.ElementTree — The ElementTree XML API.“ Online; abgerufen am 21.08.2025, besucht am 21. Aug. 2025. Adresse: <https://docs.python.org/3/library/xml.etree.elementtree.html#module-xml.etree.ElementTree>
- [7] Python Software Foundation. „xml.dom.minidom — Minimal DOM implementation.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://docs.python.org/3/library/xml.dom.minidom.html>
- [8] Python Software Foundation. „xml.sax — Support for SAX2 parsers.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://docs.python.org/3/library/xml.sax.html>
- [9] S. Richter. „lxml — XML and HTML with Python.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://lxml.de/>
- [10] Python Software Foundation. „beautifulsoup4 4.13.5.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://pypi.org/project/beautifulsoup4/>
- [11] Python Software Foundation. „xmlschema 4.1.0.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://pypi.org/project/xmlschema/>
- [12] Python Software Foundation. „defusedxml 0.7.1.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://pypi.org/project/defusedxml/>
- [13] C. Stefanescu. „untangle: Convert XML to Python objects.“ Online; abgerufen am 22.09.2025, besucht am 22. Sep. 2025. Adresse: <https://untangle.readthedocs.io/en/latest/>

- [14] E. Fuchs, *SQL: Grundlagen und Datenbankdesign*, 1. Ausgabe. Bodenheim: HERDT-Verlag, Juli 2021, ISBN: 978-3-98569-009-1.
- [15] F. Herrmann, *Datenorganisation und Datenbanken: Praxisorientierte Übungen mit MS Access 2016*. Wiesbaden: Springer Vieweg, 2018, ISBN: 978-3-658-21330-5. DOI: 10.1007/978-3-658-21331-2 Adresse: <https://doi.org/10.1007/978-3-658-21331-2>
- [16] A. Gadatsch, *Datenmodellierung: Einführung in die Entity-Relationship-Modellierung und das Relationenmodell*, 2., aktualisierte Auflage. Wiesbaden: Springer Vieweg, 2019, ISBN: 978-3-658-25729-3. DOI: 10.1007/978-3-658-25730-9 Adresse: <https://doi.org/10.1007/978-3-658-25730-9>

ii. Anhangsverzeichnis

iii. Erklärung

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

Ort und Datum

persönliche Unterschrift

(Name des Verfassers)