Ein Bild, das Text, Schrift, Logo, Design enthält.

Automatisch generierte Beschreibung

Abschlussprüfung Winter 2024 / 2025 Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Konfiguration Generator

**Konfigurationsgenerator für den Ammonit Meteo 40 Datenlogger**

Abgabedatum: Würzburg, den 15.01.2024

**Prüfungsbewerber:** Kim Plecker

Am Sportplatz 17

55276 Dienheim

**Ausbildungsbetrieb:** BFW Würzburg

Helen-Keller-Straße 6

96207 Würzburg

**Praktikumsbetrieb:**

ProfEC Ventus

Am Patentbusch 3B

26125 Oldenburg

# Inhaltsverzeichnis

Inhalt

[Inhaltsverzeichnis 1](#_Toc185322198)

[Verzeichnis der Listings 5](#_Toc185322199)

[1.1 Projektumfeld 1](#_Toc185322200)

[1.2 Projektziel 1](#_Toc185322201)

[1.3 Projektbegründung 1](#_Toc185322202)

[1.4 Projektschnittstellen 1](#_Toc185322203)

[1.5 Projektabgrenzung 1](#_Toc185322204)

[2.1 Projektphasen 2](#_Toc185322205)

[2.2 Abweichungen vom Projektantrag 2](#_Toc185322206)

[2.3 Ressourcenplanung 2](#_Toc185322207)

[2.4 Entwicklungsprozess 3](#_Toc185322208)

[3.1 Ist-Analyse 3](#_Toc185322209)

[3.2 Wirtschaftlichkeitsanalyse 3](#_Toc185322210)

[3.2.1 Make or Buy-Entscheidung 4](#_Toc185322211)

[3.2.2 Amortisationsdauer 4](#_Toc185322212)

[3.3 Nutzwertanalyse 6](#_Toc185322213)

[3.4 Anwendungsfälle 6](#_Toc185322214)

[3.5 Qualitätsanforderungen 6](#_Toc185322215)

[3.6 Lastenheft/Fachkonzept 6](#_Toc185322216)

[4.1 Zielplattform 6](#_Toc185322217)

[4.2 Architekturdesign 6](#_Toc185322218)

[4.3 Entwurf der Benutzeroberfläche 7](#_Toc185322219)

[4.4 Datenmodell 7](#_Toc185322220)

[4.5 Geschäftslogik 7](#_Toc185322221)

[4.6 Maßnahmen zur Qualitätssicherung 8](#_Toc185322222)

[4.7 Pflichtenheft/Datenverarbeitungskonzept 8](#_Toc185322223)

[5.1 Implementierung der Datenstrukturen 8](#_Toc185322224)

[5.2 Implementierung der Benutzeroberfläche 8](#_Toc185322225)

[5.3 Implementierung der Geschäftslogik 8](#_Toc185322226)

[9.1 Soll-/Ist-Vergleich 9](#_Toc185322227)

[9.2 Lessons Learned 10](#_Toc185322228)

[9.3 Ausblick 10](#_Toc185322229)

[A1 Detaillierte Zeitplanung 1](#_Toc185322230)

[A2 Lastenheft (Auszug) 2](#_Toc185322231)

[A3 Use-Case-Diagramm 3](#_Toc185322232)

[A4 Pflichtenheft (Auszug) 3](#_Toc185322233)

[A5 Datenbankmodell 5](#_Toc185322234)

[A6 Ereignisgesteuerte Prozesskette 6](#_Toc185322235)

[A7 Oberflächenentwürfe 7](#_Toc185322236)

[A8 Screenshots der Anwendung 8](#_Toc185322237)

[A9 Entwicklerdokumentation (Auszug) 10](#_Toc185322238)

[A10 Testfall und sein Aufruf auf der Konsole 11](#_Toc185322239)

[A11 Klasse: ComparedNaturalModuleInformation 12](#_Toc185322240)

[A12 Klassendiagramm 14](#_Toc185322241)

[A13 Benutzerdokumentation (Auszug) 15](#_Toc185322242)

**Abbildungsverzeichnis**

**Abbildungsverzeichnis**

[Abbildung 1: Use-Case-Diagramm iii](#_bookmark56)

[Abbildung 2: Entity-Relationship-Model v](#_bookmark59)

[Abbildung 3: Tabellenmodell vi](#_bookmark60)

[Abbildung 4: Prozess des Einlesens eines Moduls vi](#_bookmark62)

[Abbildung 5: Liste der Module mit Filtermöglichkeiten vii](#_bookmark64)

[Abbildung 6: Anzeige der Übersichtsseite einzelner Module viii](#_bookmark65)

[Abbildung 7: Anzeige und Filterung der Module nach Tags viii](#_bookmark67)

[Abbildung 8: Liste der Module mit Filtermöglichkeiten ix](#_bookmark68)

[Abbildung 9: Auszug aus der Entwicklerdokumentation mit *PHPDoc* x](#_bookmark70)

[Abbildung 10: Aufruf des Testfalls auf der Konsole xi](#_bookmark72)

[Abbildung 11: Klassendiagramm xiv](#_bookmark77)

[Abbildung 12: Auszug aus der Benutzerdokumentation xv](#_bookmark79)

**Tabellenverzeichnis**

**Tabellenverzeichnis**

[Tabelle 1: Grobe Zeitplanung 2](#_bookmark13)

[Tabelle 2: Kostenaufstellung 3](#_bookmark23)

[Tabelle 3: Entscheidungsmatrix 5](#_bookmark31)

[Tabelle 4: Soll-/Ist-Vergleich 8](#_bookmark46)

[Tabelle 5: Detaillierte Zeitplanung ii](#_bookmark53)

#### Verzeichnis der Listings

# Verzeichnis der Listings

[Listing 1: Testklasse xii](#_bookmark73)

[Listing 2: Klasse ComparedNaturalModuleInformation xiv](#_bookmark75)

**Abkürzungsverzeichnis**

**Abkürzungsverzeichnis**

API *Application Programming Interface*

CSV *Comma Separated Values*

EPK *Ereignisgesteuerte Prozesskette*

ERM *Entity Relationship Model*

GUI *Graphical User Interface*

HTML *Hypertext Markup Language*

MVC *Model View Controller*

PHP *PHP Hypertext Preprocessor*

SQL *Structured Query Language*

SVN *Subversion*

XML *Extensible Markup Language*

**Einleitung**

**1 Einleitung**

## Projektumfeld

* + - Die Firma ProfEC Ventus ist ein global agierendes Unternehmen mit Hauptsitz in Oldenburg, Niedersachsen mit ungefähr 20 Mitarbeitern. Die Firma wurde im Jahr 2012 gegründet und wächst seitdem stetig. Sie entwickelt und baut Messsysteme, mit denen für ihre Kunden Wind und Solargutachten zu potenziellen Standorten erstellt werden. ProfEC Ventus bietet Komplettlösungen an, die von der Stromversorgung über Batterien, E-Fuels (Wasserstoffbrennzellen), Solarzellen und Stromgeneratoren bis hin zu vorkonfektionierten Kabeln und selbst kalibrierten Sensoren reichen.
    - Die Messsysteme verbleiben mehrere Jahre am selben Standort, um meteorologische Daten zu erfassen. Diese Daten werden von uns ausgewertet und in ein Windgutachten überführt. Das Gutachten dient dazu, den Standort als potenziellen Wind- oder Solarpark zu evaluieren und gegenüber der Bank das Potenzial des Standortes nachzuweisen oder diesen Standort ausschließen zu können.
    - In den Messsystemen, die ProfEC Ventus herstellt und vertreibt, wird ein Datenlogger installiert. Dieser ist von zentraler Bedeutung und die Kernkomponente jedes Systems. Der Datenlogger sammelt sämtliche Daten, die von den an den verschiedenen Kanälen angeschlossenen Sensoren bereitgestellt werden. Außerdem übernimmt der Datenlogger die Kontrolllogik für verschiedenste Aufgaben im System, wie zum Beispiel das Schalten der Relais für die Sensorheizungen oder die Aktivierung des Mobilfunkrouters, um die gesammelten Daten der Außenwelt zugänglich zu machen.
    - Auftraggeber des Projektes ist der Praktikumsgeber ProfEC Ventus.

## Projektziel

* + - Das Ziel des Projektes besteht darin einen Generator zu entwickeln welcher in der Lage ist eine maßgeschneiderte Konfiguration für den Datenlogger zu generieren. Jedes neue System benötigt eine maßgeschneiderte Konfiguration da immer wieder unterschiedliche Sensorkombinationen an den Datenlogger angeschlossen werden. Diese müssen für jedes System korrekt erfasst werden. Dies soll den bisherigen Arbeitsprozess der über 2 Stunden pro System und 2 Mitarbeiter gebunden hat ersetzen, so dass die Arbeitszeit sinnvoll woanders eingesetzt werden kann.

## Projektbegründung

* + - Das Projekt zielt darauf ab, eine Zeiteinsparung von 5 Arbeitsstunden pro Woche zu erzielen und gleichzeitig Eingabefehler im Konfigurationsprozess deutlich zu reduzieren.
    - Die Hauptmotivation besteht darin, wertvolle Zeit einzusparen und diese für andere wichtige Aufgaben sinnvoll nutzen zu können. Ein zusätzlicher Vorteil ist, dass Eingabefehler durch die Automatisierung effektiv vermieden werden.

## Projektschnittstellen

* + - Das Projekt soll später in andere Projekte miteingebunden werden. Dazu benötigt es eine Schnittstelle. Ein Projekt davon wird die Firmeninterne Teststation darstellen. Weiter Projekte sind möglich aber noch nicht geplant.
    - Das Projekt wurde von der Firma ProfEC Ventus genehmigt und die erforderlichen Mittel werden zu Verfügung gestellt.
    - Die Anwender des Softwareprodukts werden vollqualifizierte Mitarbeiter, die sich in der Programmierung auskennen und über das entsprechende Know-how verfügen Programme auf der Shell auszuführen.
    - Das Ergebnis muss sowohl dem Leiter der IT-Abteilung sowie dem Leiter der Werkstatt präsentiert werden.

## Projektabgrenzung

* + - Nicht Bestandteil des Projekts werden folgende Punkte sein:

1. Die Steuerung der Modemzeiten
2. Die Steuerung wie der Datenlogger nach Außen Kommuniziert
3. Ultraschallsensoren
4. Die Steuerung der Logikschaltung
   * + Bestandteil des Projektes werden folgende Punkte sein
       1. Anemometer (Windgeschwindigkeitssensoren)
       2. Windvanes (Windrichtungssensoren)
       3. Barometer (Atmosphärendrucksensoren)
       4. Temperatur und Feuchtigkeitssensoren (Multisensor)

Diese 4 Sensortypen sind in 99% aller unserer Systeme vorhanden.

**2 Projektplanung**

## Projektphasen

* + - In welchem Zeitraum und unter welchen Rahmenbedingungen (z.B. Tagesarbeitszeit) findet das Projekt statt?
    - Verfeinerung der Zeitplanung, die bereits im Projektantrag vorgestellt wurde.
    - Das Projekt findet während der Regulären Tagesarbeitszeitstatt für diese Zeit wurde ich vom Unternehmen für die im Rahmen des Projektes anfallenden Arbeiten von sonstigen Arbeitsaufgaben freigestellt.
    - Während der Analysephase erstelle ich eine ausführliche Ist Analyse, eine Soll Analyse und eine Wirtschaftlichkeitsanalyse
    - Während der Entwurfsphase erstelle ich einen groben Datenverarbeitungsentwurf, daraus werden verschiedene Klassen erzeugt, daraus wiederum ein Klassendiagramm erstellt und damit werde ich mich entscheiden wie die Benutzerschnittstelle aussehen wird.
    - Während der Entwicklungsphase werden die jeweiligen Klassen erstellt (Sensoren, Generator) Sowie die benötigten Hilfsfunktionen und die Excel DB.
    - Anschließend folgt die Testphase in welcher ich verschiedene bestehende Konfigurationen neu generiere und mit den Ursprünglichen vergleiche sowie auf einen Datenlogger aufspiele um die Lauffähigkeit zu gewährleisten. Zusätzlich werde ich einen Integrationstest auf einigen Computern meiner Kollegen durchführen. Die Software wird auch mit fehlerhaftem Input getestet.
    - Zum Schluss wird die Dokumentation geschrieben.

**Beispiel**

[Tabelle 1](#_bookmark13) zeigt ein Beispiel für eine grobe Zeitplanung.1

Eine detailliertere Zeitplanung ist in [Tabelle 5](#_bookmark53) in Anhang [A1](#_bookmark52) zu sehen.

|  |  |
| --- | --- |
| Projektphase | Geplante Zeit (Stunden) |
| Analysephase | 8 h |
| Entwurfsphase | 11 h |
| Entwicklungsphase | 40 h |
| Testphase | 7 h |
| Dokumentation | 14 h |
| Gesamt | 80 h |

**Tabelle 1: Grobe Zeitplanung**

## Abweichungen vom Projektantrag

* + - Abweichung vom Projektantrag: Änderung des Zeitlichen Rahmens.
* Begründung: Mein Arbeitsmittel ist ausgefallen (Laptop) 1 Tag vor Projektstart. Ein Ersatz konnte erst eine Woche später geliefert werden.
* Auswirkungen: Projektstart verschiebt sich um 1 Woche. Keine Auswirkungen auf interne Prozesse oder Beeinträchtigung von Folgeprojekten.
* Neuer zeitlicher Rahmen: 09.12.2024 – 10.01.2025

## Ressourcenplanung

* + - Benötigte Hardware: Laptop, Datenlogger (Ammonit Meteo 40M), PC-Bildschirm, Maus, Tastatur.
    - Benötigte Software: Word, Excel, Anaconda (Python), Visual Studio Code
    - Benötigtes Umfeld: Eingerichteter Büroarbeitsplatz (Schreibtisch, Schreibtischstuhl)
    - Beratender Mitarbeiter: Ausbilder

## Entwicklungsprozess

* + - Der Entwicklungsprozess wird dem Wasserfallmodell folgen.
    - Die Phasen des Klassischen Wasserfallmodells:

1. Analyse
2. Design
3. Implementierung
4. Test
5. Inbetriebnahme

**3 Analysephase**

## Ist-Analyse

* Bisher gibt es kein Programm, welches die Konfiguration des Datenloggers übernimmt. Bei jedem Projekt wird der Datenlogger manuell von Hand konfiguriert. Da jedes Projekt unterschiedlich ist muss diese Konfiguration bei jedem Datenlogger individuell vorgenommen werden. Dies benötigt erhebliche Konzentration und eine hohe Aufmerksamkeit. Der Aktuelle Prozess in der Firma sieht wie Folgt aus:

1. Datenlogger mit Strom versorgen und ins Netzwerk einbinden (Über LAN)
2. Die im Datenlogger intern gehostete Webseite aufrufen
3. Jeden Sensor (bis zu 20 Stück (Der Datenlogger könnte mehr, aber das ist sehr unüblich im Tagesgeschäft)) einzeln über eine Maske auswählen.
4. Sämtliche Daten, aus dem Herstellungsplan eingeben, sowie die Zertifikate (4-8 Seiten pro Sensor) durchsuchen und die erforderlichen Daten (Slope und Offset) in die Maske übertragen
5. Validierung der eingegebenen Daten durch einen zweiten Mitarbeiter.

Der Wunsch des Chefs sowie der Abteilungen ist Schritt 3 – 5 zu Automatisieren

* Das Ziel wird sein ein Programm zu erstellen welches automatisiert den Herstellungsplan einliest, sowie die PDFs ausliest. Mit diesen Daten wird eine Konfigurationsdatei erstellt, welche dann in den Datenlogger hochgeladen werden kann.

## Wirtschaftlichkeitsanalyse

* Auflistung der Fixkosten:

1. Arbeitszeitkosten Azubi: 0€ (Unbezahltes Praktikum)
2. Büroraum (inkl. Nebenkosten, Reinigung, etc.): 24,83€ pro Tag
3. Quelle [Der große Preisvergleich: Coworking vs. Büro? | SleevesUp!](https://www.sleevesup.de/magazin/preisvergleich-coworking-oder-buro/)
4. Zeitkosten des beratenden Mitarbeiters: 10 Stunden a 100€ = 1000€
5. Kostenpflichtige Software: Microsoft 365 Business Standard: 11,70€/Monat
6. Schulung und Einführung für Mitarbeiter: 3 Stunden a 40€
7. Abnahme durch Fachabteilung: 1 Stunde a 80€

Gesamt: **1.763€**

* Auflistung der laufenden Kosten:

1. Einfügen eines bereits als Klasse angelegten Sensors in das Programm:

2 Stunden a 40€ gesamt 80€ (Häufigkeit: Nicht absehbar)

1. Einfügen eines noch nicht als Klasse angelegten Sensors in das Programm:

5 Stunden a 40€ gesamt 200€ (Häufigkeit: Nicht absehbar)

1. Wartung und Pflege des Programms:

2 Stunden / Monat a 40€ gesamt 80€

Gesamt: 80€ + (schätzungsweise jeweils 1 Sensor aus Punkt 1 und 2 pro Monat)

Entspricht jährlichen Kosten von **3.440€** pro Jahr.

* Auflistung der Gesamtkosten

Fixkosten: 1.763€

Absehbare laufende Kosten pro Jahr: 3440€

Gesamtkosten im Zeitraum von Jahr 1: 5.203€

Jahreskosten ab Jahr 2: 3.440€

* Auflistung der Ersparnisse durch Projektumsetzung

1. Zeitersparnis 5 Stunden pro Woche (Bisher 6 Stunden, nach Projektumsetzung 1 Stunde) 5\*52= 260 Arbeitsstunden im Jahr
2. Kostenersparnis 5 Stunden pro Woche a 40€ = 200€ / Woche \* 52 = 10400€
3. Gesamt: **10400€ sowie 208 Arbeitsstunden pro Jahr**

### Make or Buy-Entscheidung

* + - * Bisher gibt es keine vergleichbare Software, da es sich hierbei um eine Speziallösung innerhalb des Unternehmens handelt. Daher muss es selbst programmiert werden.

### Amortisationsdauer

* Amortisationsdauer

Gesamtkosten Jahr 1 : 5.203€

Gesamtersparnis pro Jahr : 10.400€

10400/5203 ~ 2

Das Projekt amortisiert sich nach knapp **einem halben Jahr.**

## Da die Instandhaltungskosten pro weiterem Jahr 3.440€ betragen und die Ersparnis im Jahr weiterhin 10.400€ beträgt, bleibt das Projekt auf lange Sicht ebenfalls rentabel. Sollten in Zukunft mehr Aufträge reinkommen erhöht sich die Ersparnis nochmals während die Laufenden kosten gleich bleiben.

* Die nicht Monetäre Ersparnis liegt bei **208 Arbeitsstunden** pro Jahr

## Anwendungsfälle

* Das Projekt soll dem Nutzer die Schnittstelle bieten eine Konfigurationsdatei automatisiert erstellen zu lassen. Dabei werden die PDFs automatisiert durchsucht und die gefundenen Werte direkt eingesetzt. Der Admin, der das System wartet und pflegt soll die Möglichkeiten haben neue Sensoren einzufügen sowie eventuelle Fehler zu beheben.
* Der Generator wird von 2 Personengruppen bedient. Einmal der User sowie der Admin.

Der User bekommt eine Schnittstelle über welcher er das Programm ausführen kann sowie einen Pfad übergeben kann.

Der Admin bekommt ebenfalls die Möglichkeit das Programm auszuführen sowie das System zu erweitern, zu pflegen, Fehler zu beheben.

**Beispiel**

Ein Beispiel für ein Use-Case-Diagramm findet sich im Anhang [A3](#_bookmark55).

## Qualitätsanforderungen

* Es sollen die Qualitätsmerkmale nach ISO 9126 erreicht werden
  + Änderbarkeit – Leicht zu ändern, zu modifizieren sowie zu Testen
  + Effizienz – Es soll möglichst effizient arbeiten
  + Übertragbarkeit – Leicht auf fremden Rechnern ausführbar sein
  + Zuverlässigkeit – Fehlertoleranz möglichst klein halten
  + Funktionalität – Die erzeugten Ergebnisse müssen verwertbar sein
  + Benutzbarkeit – Es soll gerne und leicht von den Usern bedient werden

**4 Entwurfsphase**

## Zielplattform

* + - Das Projekt wird in Python umgesetzt sowie Excel als Datenbankersatz genutzt. Die Software wird auf allen Windows Rechnern sowie Linux Rechnern welche Python fähig sind einsetzbar sein.

Die Kriterien dafür sind:

* 1. Einfache Bedienbarkeit: Das Programm soll auch von projektfremden Personen problemlos ausführbar sein.
  2. Wartbarkeit: Der Code muss für andere Entwickler leicht verständlich und wartbar sein.
  3. Erweiterbarkeit: Das System soll so gestaltet sein, dass es von Entwicklern unkompliziert erweitert werden kann

Im Unternehmen wird primär Python für inhouse Lösungen eingesetzt. Da das Unternehmen noch recht klein ist wird momentan noch keine größere Datenbanklösung wie z.b. SQLite eingesetzt.

## Architekturdesign

* + - Das Projekt wird primär Klassenbasiert umgesetzt um die Wartbarkeit, Erweiterbarkeit sowie die Lesbarkeit des Codes zu gewährleisten.
    - Ich habe mich bewusst für einen klassenbasierten Ansatz entschieden, im Gegensatz zur prozeduralen Programmierung, da dieser klare Abgrenzungen zwischen den Komponenten schafft. Dies erleichtert die Identifikation und Behebung von Schwachstellen und Fehlern sowie die zukünftige Erweiterung des Programms.
    - Folgende Programmierparadigmen sollen berücksichtigt werden:
      * Serpation of Concerns
      * Divide and conquer
      * Naming convention

## Entwurf der Benutzeroberfläche

* + - Da die Software nur von vollqualifiziertem und geschultem Personal eingesetzt werden wird habe ich mich für eine Konstruktor Schnittstelle entschieden. Hier wird der Konstruktor des Generators aufgerufen und ein Dateipfad übergeben. Dies löst den gesamten Prozess aus und geniert die Konfiguration. So das in weitern noch nicht geplanten Projekten nur das Projekt importiert werden muss / kann.

**Beispiel**

Einen Screenshot des aufrufen, finden sich im Anhang [A7](#_bookmark63).

## Datenmodell

* + - Entwurf/Beschreibung der Datenstrukturen (z.B. ERM und/oder Tabellenmodell, XML- Schemas) mit kurzer Beschreibung der wichtigsten (!) verwendeten Entitäten.
    - Es werden folgende Klassen verwendet
      * Sensoren: SensorBase, Anemometer, Barometer, Windvane, Temperature\_and\_Humidity
      * Generator: Generator
    - Ein Dictionary welches Statische Informationen bereit hält
    - Sowie notwendige Hilfs Funktionen
      * Excel Reader, PDF-Reader and Filter
    - In der Exceldatenbank werden folgende Daten gespeichert
      * Type\_no, type, model, eval\_type, model\_id, channel\_type, mess\_typ, slope, offset, formula, formula\_params, protocol, order\_num, adc\_range\_u, sensor\_type, statictics, counter\_evals, url
    - Die Sensorklassen dienen dazu, einzelne Sensoren aus der Produktion programmtechnisch abzubilden und als Objekte zu erstellen. Für jeden Sensor, der in einem Projekt für einen bestimmten Kunden genutzt wird, wird ein eigenes Objekt erstellt und dem Generator übergeben. Der Generator verwendet diese Objekte zusammen mit den, in Ihm hinterlegten benötigten Daten, um eine Konfigurationsdatei zu erstellen. Diese Konfigurationsdatei kann anschließend auf den Ammoniten aufgespielt werden.

## Geschäftslogik

* + - Modellierung und Beschreibung der wichtigsten (!) Bereiche der Geschäftslogik (z.B. mit Komponenten-, Klassen-, Sequenz-, Datenflussdiagramm, Programmablaufplan, Struktogramm, EPK).
    - Wie wird die erstellte Anwendung in den Arbeitsfluss des Unternehmens integriert?

**Beispiel**

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen unterei- nander darstellt, kann im Anhang [A12](#_bookmark76) eingesehen werden.

Die EPK in Anhang [A6](#_bookmark61) zeigt den grundsätzlichen Ablauf beim Einlesen eines Moduls.

## Maßnahmen zur Qualitätssicherung

* + - Welche Maßnahmen werden ergriffen, um die Qualität des Projektergebnisses (siehe Kapitel [3.5](#_bookmark26)) zu sichern (z.B. automatische Tests, Anwendertests)?
    - Ggfs. Definition von Testfällen und deren Durchführung (durch Programme/Benutzer).

## Pflichtenheft/Datenverarbeitungskonzept

* + - Sensoren Klassen: Anemometer, Barometer, Umgebungstemperatur und relative Luftfeuchtigkeit, Windfahne, Barometer
    - Generator Klasse: Generator
    - Unterstützende Klassen: PDF Extractor, Dantenbanklader, Herstellungsplanleser
    1. Der User erzeugt eine Instanz der Generator Klasse und übergibt seinen Dateipfad im Konstruktor.
    2. Der Generator erstellt eine Instanz des Herstellungsplanlesers.
    3. Diese lädt den Herstellungsplan und übergibt die Daten dem Generator.
    4. Die Daten werden gefiltert und die verschiedenen Sensor Objekte erstellt sowie die PDFs ausgelesen und dem Generator hinzugefügt.
    5. Der Generator enthält nun alle Daten des Sensors sowie der PDFs und der Datenbank und geniert nun die Konfigurationsdatei.
    6. Er erstellt einen neuen Ordner im Stammverzeichnis des Projektes und legt dort die neue Konfigurationsdatei im INI Format ab.

**Beispiel**

Ein Beispiel für das auf dem Lastenheft (siehe Kapitel [3.6](#_bookmark27)) aufbauende Pflichtenheft ist im Anhang [A4](#_bookmark57) zu finden.

**5 Implementierungsphase**

## Implementierung der Datenstrukturen

* + - In der ExcelDB wurden folgende Werte angelegt: type\_no, type, model, eval\_type, model\_id, channel\_type, mess\_typ, slope, offset (Falls diese Werte nicht in einem Zertifikat gefunden werden können sind hier Default Werte hinterlegt), formula, formula\_params, protocol, order\_num, adc\_range\_u, sensor\_type, statistics, counter\_eval sowie url
    - Diese Werte bilden das Grundgerüst. In einer SQL-Datenbank würde die Tabelle in mehrere Tabellen aufgeteilt werden, um die dritte Normalform zu gewährleisten. Da es sich jedoch um eine Excel-Datenbank mit einer überschaubaren Anzahl von Datensätzen handelt und SQL-Datenbanken in der Firma bislang nicht genutzt werden, fiel die bewusste Entscheidung zugunsten einer Excel-Liste.

## Implementierung der Benutzeroberfläche

* + - Screenshots der Anwendung
    - Ich habe mich bewusst für einen konsolenbasierten Aufruf des Konstruktors der Generatorklasse entschieden. Dies ermöglicht es dem Nutzer, das Programm in nur einem Schritt auszuführen.

**Beispiel**

Screenshots der Anwendung in der Entwicklungsphase mit Dummy-Daten befinden sich im Anhang [A8](#_bookmark66).

## Implementierung der Geschäftslogik

* + - Beschreibung des Vorgehens bei der Umsetzung/Programmierung der entworfenen Anwendung.
    - Ggfs. interessante Funktionen/Algorithmen im Detail vorstellen, verwendete Entwurfs- muster zeigen.
    - Quelltextbeispiele zeigen.
    - Hinweis: Es wird nicht ein lauffähiges Programm bewertet, sondern die Projektdurch- führung. Dennoch würde ich immer Quelltextausschnitte zeigen, da sonst Zweifel an der tatsächlichen Leistung des Prüflings aufkommen können.

**Abnahmephase**

**Beispiel**

Die Klasse ComparedNaturalModuleInformation findet sich im Anhang [A11](#_bookmark74).

**6 Abnahmephase**

* + - Welche Tests (z.B. Unit-, Integrations-, Systemtests) wurden durchgeführt und welche Ergebnisse haben sie geliefert (z.B. Logs von Unit Tests, Testprotokolle der Anwen- der)?
    - Wurde die Anwendung offiziell abgenommen?

**Beispiel**

Ein Auszug eines Unit Tests befindet sich im Anhang [A10](#_bookmark71). Dort ist auch der Aufruf des Tests auf der Konsole des Webservers zu sehen.

**7 Einführungsphase**

* + - Welche Schritte waren zum Deployment der Anwendung nötig und wie wurden sie durchgeführt (automatisiert/manuell)?
    - Wurden Ggfs. Altdaten migriert und wenn ja, wie?
    - Wurden Benutzerschulungen durchgeführt und wenn ja, Wie wurden sie vorbereitet?

**8 Dokumentation**

* + - Wie wurde die Anwendung für die Benutzer/Administratoren/Entwickler dokumentiert (z.B. Benutzerhandbuch, API-Dokumentation)?
    - Hinweis: Je nach Zielgruppe gelten bestimmte Anforderungen für die Dokumentation (z.B. keine IT-Fachbegriffe in einer Anwenderdokumentation verwenden, aber auf je- den Fall in einer Dokumentation für den IT-Bereich).

**Beispiel**

Ein Ausschnitt aus der erstellten Benutzerdokumentation befindet sich im Anhang [A13](#_bookmark78).

Die Entwicklerdokumentation wurde mittels *PHPDoc* automatisch generiert. Ein beispielhafter Auszug aus der Dokumentation einer Klasse findet sich im Anhang [A9](#_bookmark69).

**9 Fazit**

## Soll-/Ist-Vergleich

* + - Wurde das Projektziel erreicht und wenn nein, warum nicht?
    - Ist der Auftraggeber mit dem Projektergebnis zufrieden und wenn nein, warum nicht?
    - Wurde die Projektplanung (Zeit, Kosten, Personal, Sachmittel) eingehalten oder haben sich Abweichungen ergeben und wenn ja, warum?
    - Hinweis: Die Projektplanung muss nicht strikt eingehalten werden. Vielmehr sind Ab- weichungen sogar als normal anzusehen. Sie müssen nur vernünftig begründet wer- den (z.B. durch Änderungen an den Anforderungen, unter-/überschätzter Aufwand).

**Beispiel (verkürzt)**

Wie in [Tabelle 4](#_bookmark46) zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehal- ten werden.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fazit** |  | | |
|  |
|  | **Phase** | **Geplant Tatsächlich** | **Differenz** |
|  | Analyse | 9 h 10 h | +1 h |
|  | Entwurf | 20 h 20 h |  |
|  | Implementierung | 30 h 27 h | -3 h |
|  | Abnahme | 1 h 1 h |  |
|  | Einführung | 1 h 1 h |  |
|  | Dokumentation | 9 h 11 h | +2 h |
|  | **Gesamt** | **70 h 70 h**  **Tabelle 4: Soll-/Ist-Vergleich** |  |

## Lessons Learned

* + - Was hat der Prüfling bei der Durchführung des Projekts gelernt (z.B. Zeitplanung, Vor- teile der eingesetzten Frameworks, Änderungen der Anforderungen)?

## Ausblick

* + - Wie wird sich das Projekt in Zukunft weiterentwickeln (z.B. geplante Erweiterungen)?

**Literaturverzeichnis**

**Literaturverzeichnis**

Grashorn, D., 2010. *Entwicklung von NatInfo – Webbasiertes Tool zur Unterstützung der Entwickler,* Vechta: s.n.

ISO/IEC 9126-1, 2001. *Software-Engineering – Qualität von Software-Produkten – Teil 1: Qualitätsmodell.* s.l.:s.n.

**Eidesstattliche Erklärung**

**Eidesstattliche Erklärung**

Ich, Der Autor, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektar- beit mit dem Thema

*Konfigurationsgenerator für den Datenlogger Ammonit Meteo 40*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Oldenburg, den 14.1.2025



Kim Christopher Plecker

**Anhang**

## A1 Detaillierte Zeitplanung

|  |  |  |  |
| --- | --- | --- | --- |
| **Analysephase** |  |  | **9 h** |
| 1. Analyse des Ist-Zustands |  | 3 h |  |
| 1.1. Fachgespräch mit der EDV-Abteilung | 1 h |  |  |
| 1.2. Prozessanalyse | 2 h |  |  |
| 2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse |  | 1 h |  |
| 3. Erstellen eines Use-Case-Diagramms |  | 2 h |  |
| 4. Erstellen des Lastenhefts mit der EDV-Abteilung |  | 3 h |  |
| **Entwurfsphase** |  |  | **20 h** |
| 1. Prozessentwurf |  | 3 h |  |
| 2. Datenbankentwurf |  | 3 h |  |
| 2.1. ER-Modell erstellen | 2 h |  |  |
| 2.2. Konkretes Tabellenmodell erstellen | 1 h |  |  |
| 3. Erstellen von Datenverarbeitungskonzepten |  | 4 h |  |
| 3.1. Verarbeitung der CSV-Daten | 1 h |  |  |
| 3.2. Verarbeitung der SVN-Daten | 1 h |  |  |
| 3.3. Verarbeitung der Sourcen der Programme | 2 h |  |  |
| 4. Benutzeroberflächen entwerfen und abstimmen |  | 2 h |  |
| 5. Erstellen eines UML-Komponentendiagramms der Anwendung |  | 4 h |  |
| 6. Erstellen des Pflichtenhefts |  | 4 h |  |
| **Implementierungsphase** |  |  | **30 h** |
| 1. Anlegen der Datenbank |  | 1 h |  |
| 2. Umsetzung der HTML-Oberflächen und Stylesheets |  | 5 h |  |
| 3. Programmierung der PHP-Module für die Funktionen |  | 23 h |  |
| 3.1. Import der Modulinformationen aus CSV-Dateien | 2 h |  |  |
| 3.2. Parsen der Modulquelltexte | 3 h |  |  |
| 3.3. Import der SVN-Daten | 2 h |  |  |
| 3.4. Vergleichen zweier Umgebungen | 4 h |  |  |
| 3.5. Abrufen der von einem zu wählenden Benutzer geänderten Module | 3 h |  |  |
| 3.6. Erstellen einer Liste der Module unter unterschiedlichen Aspekten | 5 h |  |  |
| 3.7. Anzeigen einer Liste mit den Modulen und geparsten Metadaten | 3 h |  |  |
| 3.8. Erstellen einer Übersichtsseite für ein einzelnes Modul | 1 h |  |  |
| 4. Nächtlichen Batchjob einrichten |  | 1 h |  |

|  |  |  |
| --- | --- | --- |
| **Abnahmetest der Fachabteilung** |  | **1 h** |
| 1. Abnahmetest der Fachabteilung | 1 h |  |
| **Einführungsphase** |  | **1 h** |
| 1. Einführung/Benutzerschulung | 1 h |  |
| **Erstellen der Dokumentation** |  | **9 h** |
| 1. Erstellen der Benutzerdokumentation | 2 h |  |

|  |  |  |
| --- | --- | --- |
| 2. Erstellen der Projektdokumentation |  | 6 h |
| 3. Programmdokumentation |  | 1 h |
| 3.1. Generierung durch *PHPdoc* | 1 h |  |
| **Gesamt** |  | **70 h** |

**Tabelle 5: Detaillierte Zeitplanung**

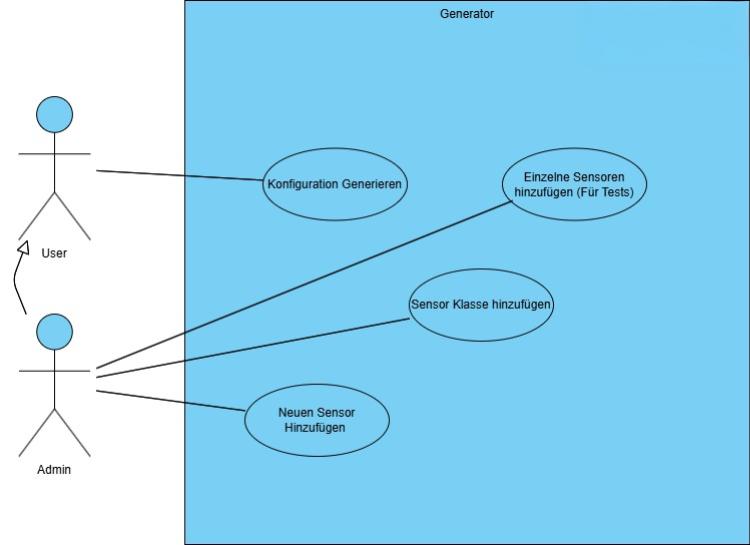
## A2 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

**Die Anwendung muss folgende Anforderungen erfüllen.**

1. Verarbeitung der Moduldaten
   1. Die Anwendung muss die von Subversion und einem externen Programm bereitgestell- ten Informationen (z.B. Source-Benutzer, -Datum, Hash) verarbeiten.
   2. Auslesen der Beschreibung und der Stichwörter aus dem Sourcecode.
2. Darstellung der Daten
   1. Die Anwendung muss eine Liste aller Module erzeugen inkl. Source-Benutzer und - Datum, letztem Commit-Benutzer und -Datum für alle drei Umgebungen.
   2. Verknüpfen der Module mit externen Tools wie z.B. Wiki-Einträgen zu den Modulen oder dem Sourcecode in Subversion.
   3. Die Sourcen der Umgebungen müssen verglichen und eine schnelle Übersicht zur Ein- haltung des allgemeinen Entwicklungsprozesses gegeben werden.
   4. Dieser Vergleich muss auf die von einem bestimmten Benutzer bearbeiteten Module eingeschränkt werden können.
   5. Die Anwendung muss in dieser Liste auch Module anzeigen, die nach einer Bearbei- tung durch den gesuchten Benutzer durch jemand anderen bearbeitet wurden.
   6. Abweichungen sollen kenntlich gemacht werden.
   7. Anzeigen einer Übersichtsseite für ein Modul mit allen relevanten Informationen zu die- sem.
3. Sonstige Anforderungen
   1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software über einen Webbrowser im Intranet erreichbar sein.
   2. Die Daten der Anwendung müssen jede Nacht bzw. nach jedem SVN-Commit automa- tisch aktualisiert werden.
   3. Es muss ermittelt werden, ob Änderungen auf der Produktionsumgebung vorgenom- men wurden, die nicht von einer anderen Umgebung kopiert wurden. Diese Modulliste soll als Mahnung per E-Mail an alle Entwickler geschickt werden (Peer Pressure).
   4. Die Anwendung soll jederzeit erreichbar sein.
   5. Da sich die Entwickler auf die Anwendung verlassen, muss diese korrekte Daten liefern und darf keinen Interpretationsspielraum lassen.
   6. Die Anwendung muss so flexibel sein, dass sie bei Änderungen im Entwicklungspro- zess einfach angepasst werden kann.

## A3 Use-Case-Diagramm

****

**Abbildung 1: Use-Case-Diagramm**

## A4 Pflichtenheft (Auszug)

**Zielbestimmung**

1. Musskriterien
   1. Modul-Liste: Zeigt eine filterbare Liste der Module mit den dazugehörigen Kerninforma- tionen sowie Symbolen zur Einhaltung des Entwicklungsprozesses an
      * In der Liste wird der Name, die Bibliothek und Daten zum Source und Kompilat eines Moduls angezeigt.
      * Ebenfalls wird der Status des Moduls hinsichtlich Source und Kompilat angezeigt. Dazu gibt es unterschiedliche Status-Zeichen, welche symbolisieren in wie weit der Entwicklungsprozess eingehalten wurde bzw. welche Schritte als nächstes getan werden müssen. So gibt es z. B. Zeichen für das Einhalten oder Verletzen des Pro- zesses oder den Hinweis auf den nächsten zu tätigenden Schritt.
      * Weiterhin werden die Benutzer und Zeitpunkte der aktuellen Version der Sourcen und Kompilate angezeigt. Dazu kann vorher ausgewählt werden, von welcher Umgebung diese Daten gelesen werden sollen.
      * Es kann eine Filterung nach allen angezeigten Daten vorgenommen werden. Die Da- ten zu den Sourcen sind historisiert. Durch die Filterung ist es möglich, auch Module zu finden, die in der Zwischenzeit schon von einem anderen Benutzer editiert wurden.
   2. Tag-Liste: Bietet die Möglichkeit die Module anhand von Tags zu filtern.
      * Es sollen die Tags angezeigt werden, nach denen bereits gefiltert wird und die, die noch der Filterung hinzugefügt werden könnten, ohne dass die Ergebnisliste leer wird.
      * Zusätzlich sollen die Module angezeigt werden, die den Filterkriterien entsprechen. Sollten die Filterkriterien leer sein, werden nur die Module angezeigt, welche mit ei- nem Tag versehen sind.
   3. Import der Moduldaten aus einer bereitgestellten CSV-Datei
      * Es wird täglich eine Datei mit den Daten der aktuellen Module erstellt. Diese Datei wird (durch einen Cronjob) automatisch nachts importiert.
      * Dabei wird für jedes importierte Modul ein Zeitstempel aktualisiert, damit festgestellt werden kann, wenn ein Modul gelöscht wurde.
      * Die Datei enthält die Namen der Umgebung, der Bibliothek und des Moduls, den Pro- grammtyp, den Benutzer und Zeitpunkt des Sourcecodes sowie des Kompilats und den Hash des Sourcecodes.
      * Sollte sich ein Modul verändert haben, werden die entsprechenden Daten in der Da- tenbank aktualisiert. Die Veränderungen am Source werden dabei aber nicht ersetzt, sondern historisiert.
   4. Import der Informationen aus Subversion (SVN). Durch einen „post-commit-hook“ wird nach jedem Einchecken eines Moduls ein PHP-Script auf der Konsole aufgerufen, wel- ches die Informationen, die vom SVN-Kommandozeilentool geliefert werden, an NatInfo übergibt.
   5. Parsen der Sourcen
      * Die Sourcen der Entwicklungsumgebung werden nach Tags, Links zu Artikeln im Wiki und Programmbeschreibungen durchsucht.
      * Diese Daten werden dann entsprechend angelegt, aktualisiert oder nicht mehr ge- setzte Tags/Wikiartikel entfernt.
   6. Sonstiges
      * Das Programm läuft als Webanwendung im Intranet.
      * Die Anwendung soll möglichst leicht erweiterbar sein und auch von anderen Entwick- lungsprozessen ausgehen können.
      * Eine Konfiguration soll möglichst in zentralen Konfigurationsdateien erfolgen.

**Produkteinsatz**

1. Anwendungsbereiche
   1. Die Webanwendung dient als Anlaufstelle für die Entwicklung. Dort sind alle Informati- onen für die Module an einer Stelle gesammelt. Vorher getrennte Anwendungen werden ersetzt bzw. verlinkt.
2. Zielgruppen
   1. NatInfo wird lediglich von den Natural-Entwicklern in der EDV-Abteilung genutzt.
3. Betriebsbedingungen
   1. Die nötigen Betriebsbedingungen, also der Webserver, die Datenbank, die Versions- verwaltung, das Wiki und der nächtliche Export sind bereits vorhanden und konfiguriert. Durch einen täglichen Cronjob werden entsprechende Daten aktualisiert, die Weban- wendung ist jederzeit aus dem Intranet heraus erreichbar.

## A5 Datenbankmodell

Ein Bild, das Diagramm, Entwurf, Plan, technische Zeichnung enthält.

Automatisch generierte Beschreibung

**Abbildung 2: Entity-Relationship-Model**

Ein Bild, das Text, Diagramm, Schrift, Quittung enthält.

Automatisch generierte Beschreibung

**Abbildung 3: Tabellenmodell**

## A6 Ereignisgesteuerte Prozesskette

Ein Bild, das gelb enthält.

Automatisch generierte Beschreibung

**Abbildung 4: Prozess des Einlesens eines Moduls**

## A7 Oberflächenentwürfe

Ein Bild, das Text, Screenshot, Zahl, Software enthält.

Automatisch generierte Beschreibung

**Abbildung 5: Liste der Module mit Filtermöglichkeiten**

Ein Bild, das Text, Screenshot, Software, Computer enthält.

Automatisch generierte Beschreibung

**Abbildung 6: Anzeige der Übersichtsseite einzelner Module**

## A8 Screenshots der Anwendung

Ein Bild, das Text, Screenshot, Website, Webseite enthält.

Automatisch generierte Beschreibung

**Abbildung 7: Anzeige und Filterung der Module nach Tags**

Ein Bild, das Text, Screenshot, Website, Webseite enthält.

Automatisch generierte Beschreibung

**Abbildung 8: Liste der Module mit Filtermöglichkeiten**

## A9 Entwicklerdokumentation (Auszug)

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

**Abbildung 9: Auszug aus der Entwicklerdokumentation mit *PHPDoc***

## A10 Testfall und sein Aufruf auf der Konsole

Ein Bild, das Text, Screenshot, Computer, Software enthält.

Automatisch generierte Beschreibung

**Abbildung 10: Aufruf des Testfalls auf der Konsole**

$t->comment('Empty Information');

$emptyComparedInformation = new ComparedNaturalModuleInformation(array());

$t->is($emptyComparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::EMPTY\_SIGN, 'Has no catalog sign');

$t->is($emptyComparedInformation->getSourceSign(), ComparedNaturalModuleInformation::SIGN\_CREATE, 'Source has to be created');

$t->comment('Perfect Module');

$criteria = new Criteria();

$criteria->add(NaturalmodulenamePeer::NAME, 'SMTAB');

$moduleName = NaturalmodulenamePeer::doSelectOne($criteria);

$t->is($moduleName->getName(), 'SMTAB', 'Right modulename selected');

$comparedInformation = $moduleName->loadNaturalModuleInformation();

$t->is($comparedInformation->getSourceSign(), ComparedNaturalModuleInformation::SIGN\_OK, 'Source sign shines global');

$t->is($comparedInformation->getCatalogSign(), ComparedNaturalModuleInformation::SIGN\_OK, 'Catalog sign shines global');

$infos = $comparedInformation->getNaturalModuleInformations(); foreach($infos as $info) {

$env = $info->getEnvironmentName();

$t->is($info->getSourceSign(), ComparedNaturalModuleInformation::SIGN\_OK, 'Source sign shines at ' . $env);

if($env != 'SVNENTW') {

$t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::SIGN\_OK, 'Catalog sign shines at ' . $info->getEnvironmentName());

} else {

$t->is($info->getCatalogSign(), ComparedNaturalModuleInformation::EMPTY\_SIGN, 'Catalog sign is empty at ' . $info->getEnvironmentName());

}

}

**Listing 1: Testklasse**

## A11 Klasse: ComparedNaturalModuleInformation

Kommentare und simple Getter/Setter werden nicht gezeigt.

class ComparedNaturalModuleInformation { const EMPTY\_SIGN = 0;

...

const SIGN\_ERROR = 5;

private $naturalModuleInformations = array(); public static function environments() {

return array("ENTW", "SVNENTW", "QS", "PROD");

}

public static function signOrder() {

return array(self::SIGN\_ERROR, self::SIGN\_NEXT\_STEP, self::SIGN\_CREATE\_AND\_NEXT\_STEP, self::SIGN\_CREATE,

self::SIGN\_OK);

}

public function construct(array $naturalInformations) {

$this->allocateModulesToEnvironments($naturalInformations);

$this->allocateEmptyModulesToMissingEnvironments();

$this->determineSourceSignsForAllEnvironments();

}

private function allocateModulesToEnvironments(array

$naturalInformations) {

foreach ($naturalInformations as $naturalInformation) {

$env = $naturalInformation->getEnvironmentName(); if(in\_array($env, self::environments())) {

$this->naturalModuleInformations[array\_search($env, self::environments())] = $naturalInformation;

}

}

}

private function allocateEmptyModulesToMissingEnvironments() { if(array\_key\_exists(0, $this->naturalModuleInformations)) {

$this->naturalModuleInformations[0]-

>setSourceSign(self::SIGN\_OK);

}

for($i = 0;$i < count(self::environments());$i++) { if(!array\_key\_exists($i, $this-

>naturalModuleInformations)) {

$environments = self::environments();

$this->naturalModuleInformations[$i] = new EmptyNaturalModuleInformation($environments[$i]);

$this->naturalModuleInformations[$i]-

>setSourceSign(self::SIGN\_CREATE);

}

}

}

private function containsSourceSign($sign) {

foreach($this->naturalModuleInformations as $information) { if($information->getSourceSign() == $sign) {

return true;

}

}

return false;

}

private function containsCatalogSign($sign) {

foreach($this->naturalModuleInformations as $information) { if($information->getCatalogSign() == $sign) {

return true;

}

}

return false;

}

}

**Listing 2: Klasse ComparedNaturalModuleInformation**

## A12 Klassendiagramm

Ein Bild, das Text, Diagramm, parallel, Plan enthält.

Automatisch generierte Beschreibung

**Abbildung 11: Klassendiagramm**

## A13 Benutzerdokumentation (Auszug)

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

**Abbildung 12: Auszug aus der Benutzerdokumentation**