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

Prüflingsnummer:

IHK Nummer:

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 2](#_Toc187391827)

[**1.1** Projektumfeld 4](#_Toc187391828)

[**1.2** Projektziel 4](#_Toc187391829)

[**1.3** Projektbegründung 4](#_Toc187391830)

[**1.4** Projektschnittstellen 4](#_Toc187391831)

[**1.5** Projektabgrenzung 5](#_Toc187391832)

[**2.1** Projektphasen 5](#_Toc187391833)

[**2.2** Abweichungen vom Projektantrag 6](#_Toc187391834)

[**2.3** Ressourcenplanung 6](#_Toc187391835)

[**2.4** Entwicklungsprozess 6](#_Toc187391836)

[**3.1** Ist-Analyse 6](#_Toc187391837)

[**3.2** Wirtschaftlichkeitsanalyse 7](#_Toc187391838)

[3.2.1 Make or Buy-Entscheidung 8](#_Toc187391839)

[3.2.2 Amortisationsdauer 8](#_Toc187391840)

[**3.3** Anwendungsfälle 8](#_Toc187391841)

[**3.4** Qualitätsanforderungen 8](#_Toc187391842)

[**4.1** Zielplattform 9](#_Toc187391843)

[**4.2** Architekturdesign 9](#_Toc187391844)

[**4.3** Entwurf der Benutzeroberfläche 10](#_Toc187391845)

[**4.4** Datenmodell 10](#_Toc187391846)

[**4.5** Geschäftslogik 10](#_Toc187391847)

[**4.6** Maßnahmen zur Qualitätssicherung 11](#_Toc187391848)

[**4.7** Datenverarbeitungskonzept 11](#_Toc187391849)

[**5.1** Implementierung der Datenstrukturen 12](#_Toc187391850)

[**5.2** Implementierung der Benutzeroberfläche 12](#_Toc187391851)

[**5.3** Implementierung der Geschäftslogik 12](#_Toc187391852)

[**9.1** Soll-/Ist-Vergleich 15](#_Toc187391853)

[**9.2** Lessons Learned 15](#_Toc187391854)

[**9.3** Ausblick 16](#_Toc187391855)

[A1 Detaillierte Zeitplanung 17](#_Toc187391856)

[A2 Use-Case-Diagramm 18](#_Toc187391857)

[A3 Ereignisgesteuerte Prozesskette 19](#_Toc187391858)

[A4 Screenshots der Anwendung 19](#_Toc187391859)

[A5 Entwickler Doku 22](#_Toc187391860)

[A6 Klasse: SensorBase 22](#_Toc187391861)

[A7 Klassendiagramm 25](#_Toc187391862)

#### Abbildungsverzeichnis

**Abbildungsverzeichnis**

Abbildung 1: Use-Case-Diagramm i

[Abbildung 2: EPK des Generators ii](#_bookmark59)

Abbildung 3: Aufruf des Programms über Batch Datei iii

Abbildung 4: Ergebnis des Tests (Falscher Input) iv

Abbildung 5: Erfolgreiche Ausführung des Programms v

Abbildung 6: Konfiguration auf Ammoniten hochladen vi

Abbildung 7: Entwickler Doku vii

Abbildung 8: Klassendiagramm viii

#### Tabellenverzeichnis

**Tabellenverzeichnis**

Tabelle 1: Grobe Zeitplanung 2

Tabelle 2: Soll-Ist-Vergleich 2

Tabelle 3: Detaillierte Zeitplanung 2

#### Abkürzungsverzeichnis

**Abkürzungsverzeichnis**

EPK *Ereignisgesteuerte Prozesskette*

GUI *Graphical User Interface*

HTML *Hypertext Markup Language*

SQL *Structured Query Language*

LAN *Local Area Network*

CMD *Command Prompt*

BAT *Batch*

**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 (Wasserstoffbrennstoffzellen), Solarzellen und Stromgeneratoren bis hin zu vorkonfektionierten Kabeln und DaKKS oder ISO 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 die Datenlogger angeschlossen werden. Diese müssen für jedes System korrekt erfasst werden. Dies soll den bisherigen Arbeitsprozess der ca. 2 Stunden pro System gedauert 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. Weitere 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

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 100% aller Systeme von ProfEC Ventus vorhanden.

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.

**2 Projektplanung**

## Projektphasen

* + - 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.
    - Nach erfolgreichen Tests wird das Programm an den Auftraggeber übergeben.
    - Während den Projektphasen wird Zeitgleich an der Projektdokumentation gearbeitet.

[Tabelle 1](#_bookmark13) zeigt ein die grobe Zeitplanung meines Projektes

Eine detailliertere Zeitplanung ist in [Tabelle 3](#_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 Auftraggebers sowie der Abteilungen ist Schritt 3 – 5 zu Automatisieren. Formatierungen!!

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/) In klammern hinter Punkt 2
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 pro Monat sowie 2x Punkt 2 pro Jahr) = 80 + ((80\*12) + (200\*2)) = 1.440€

Entspricht jährlichen Kosten von **1.440€** pro Jahr.

Auflistung der Gesamtkosten

Fixkosten: 1.763€

Absehbare laufende Kosten pro Jahr: 1.440€

Gesamtkosten im Zeitraum von Jahr 1: 3.203€

Jahresgesamtkosten ab Jahr 2: 1.440€

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€ welches 208 Arbeitsstunden pro Jahr entspricht.**

### 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 : 3.203€

Gesamtersparnis pro Jahr : 10.400€

3.203/10.400 = 0,3075 Jahre

0,3075 \*12 = 3,69 Monate

Das Projekt amortisiert sich nach knapp **3,7 Monaten.**

Da die Instandhaltungskosten pro weiterem Jahr 1.440€ 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.

## 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.

Eine Darstellung für das 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, beziehungsweise auf denen Python installiert ist, 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 (Anaconda Distribution sowie das darin integrierte Spyder) 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 einzelne Klassen sowie Ihre Funktionen können leicht wiederverwertet werden.

Folgende Programmierparadigmen sollen berücksichtigt werden:

* + - 1. Serpation of Concerns.
      2. Divide and conquer.
      3. Naming convention.

## Entwurf der Benutzeroberfläche

Die Software wird ausschließlich von qualifiziertem Personal verwendet. Daher habe ich mich für einen Aufruf über die Kommandozeile beziehungsweise über eine Batch Datei innerhalb des Projektes entschieden und somit gegen eine GUI.

Diese Batch Datei fragt den Anwender bei seiner Ausführung nach einem Dateipfad zum Herstellungsplans des Projektes. Der User muss somit nur noch den Pfad seines Herstellungsplans aus dem Firmen Projekt Ordner einfügen und Enter drücken. Zusätzlich ist möglich den Generator als Modul in zukünftige noch nicht geplante Projekte einzubinden über einen einfachen Import.

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

## Datenmodell

Es werden folgende Klassen verwendet

* + - 1. Sensoren: SensorBase, Anemometer, Barometer, Windvane, Temperature\_and\_Humidity.
      2. Generator: Generator.

Ein Dictionary welches Statische Informationen bereit hält

sowie notwendige Hilfs Funktionen für den Excel Reader, PDF-Reader und 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 Datenlogger aufgespielt werden.

Siehe Anhang A7 / Aufspielen der Konfiguration

## Geschäftslogik

* + - Es gibt 2 Möglichkeiten das Programm auszuführen:
      1. Der User wird innerhalb einer Entwicklungsumgebung (z.B. Spyder - Spyder wird hier explizit genannt, da es die gängige Umgebung innerhalb dem Unternehmen ist) den Konstruktor des Generators aufrufen und ihm einen Raw String mitgeben. In diesem Raw String ist der Pfad zu dem Herstellungsplan des jeweiligen Projektes.
      2. Über eine Batch Datei. Diese fordert den User auf den Pfad zum Herstellungsplan einzugeben. Dieser wird dann in den Generator geladen und verarbeitet.

Der Generator lädt den Herstellungsplan und überprüft ob der Richtige Datenlogger im Herstellungsplan hinterlegt ist. Ist dies der Fall werden die Sensoren aus den 4 Sensor Klassen heraus erstellt und die Sensorobjekte im Generator hinterlegt sowie die PDFs ausgelesen und die Werte den Sensoren zugeordnet. Sobald dies passiert ist lädt der Generator seine statischen Daten und fügt alles zu einer Konfiguration zusammen. Diese schreibt er Automatisch in das Projektverzeichnis in einen neuen Ordner der config heißen wird.

Danach ist es ein Einfaches die fertige Konfiguration auf den Datenlogger zu laden.

Durch die Nutzung der Cloud als Speicherort für die Projektordner wird die Implementierung des Programms erheblich vereinfacht. Alle Mitarbeitenden, die bisher eigenständig Konfigurationen erstellt haben, können nun bei neuen Projekten bequem auf den Generator zurückgreifen. Dieser speichert seine erstellten Konfigurationen direkt in der Cloud, wodurch sie für andere sofort zugänglich und nutzbar sind.

Das Programm wird auf GIT versioniert und somit zu Verfügung gestellt stehen und somit auch in Zukunft bei Neuerungen des Admins einfach zu aktualisieren sein. Eine Anleitung dafür wurde in der README hinterlegt sein.

Ein Klassendiagramm, welches die Klassen der Anwendung und deren Beziehungen untereinander darstellt, kann im Anhang [11](#_bookmark76) eingesehen werden.

Die EPK in Anhang [A6](#_bookmark61) zeigt den grundsätzlichen Ablauf des Generators.

## Maßnahmen zur Qualitätssicherung

Es wurden folgende Testfälle festgelegt:

* + - 1. Vergleich mit bestehenden Konfigurationen.
      2. Konfigurationen auf den Datenlogger laden und Testen ob der Datenlogger diese verarbeiten kann.
      3. Ein Test mit den Anwendern, um zu gewährleisten, dass eine Fehlbedienung ausgeschlossen ist.
      4. Die Installation und Ausführung des Programms erfolgt auf firmeninternen Rechnern, um mögliche Komplikationen auszuschließen.

## Datenverarbeitungskonzept

* + - Sensoren Klassen: Anemometer, Barometer, Temperature\_and\_Humidity, Windfahne, Barometer
    - Generator Klasse: Generator
    - Unterstützende Funktionen: PDF Extractor, Dantenbanklader, Herstellungsplanleser
    1. Der User / Die Batch Datei erzeugt eine Instanz der Generator Klasse und übergibt seinen Dateipfad im Konstruktor.
    2. Der Generator liest den Herstellungsplan ein sowie die Datenbank.
    3. Die Daten werden gefiltert, die verschiedenen Sensor Objekte erstellt und die PDFs ausgelesen und dem Generator hinzugefügt.
    4. Der Generator enthält nun alle Daten der Sensoren sowie der PDFs und der Datenbank und geniert nun die Konfigurationsdatei.
    5. Der Generator erstellt einen neuen Ordner im Stammverzeichnis des Projektes und legt dort die neue Konfigurationsdatei im INI Format ab.
* Nun kann der User im Stammverzeichnis des Projektes die INI Datei in den Datenlogger hochladen.

**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 // Schreiben was ich damit gemacht habe :D

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

Ich habe mich bewusst für eine Batch Datei entschieden, die im obersten Verzeichnis des Projektordners abgelegt ist. Beim Ausführen erwartet sie den Dateipfad des Herstellungsplans des Projekts. Dies ermöglicht es, den Anwendern leicht und unkompliziert das Programm auszuführen.

## Implementierung der Geschäftslogik

Als erstes stellte sich die Frage wo und wie die Daten für die jeweiligen Sensoren sowie die notwendigen Daten für den Generator hinterlegt werden. Dies führte zu einer Excel Datenbank für Sensorwerte sowie einem statischen Dict in welchem sich Daten befinden die der Generator benötigt, sich aber nicht ändern.

Danach habe ich mich daran gemacht diese Daten in mein Programm zu importieren und die Generator Klasse sowie die Sensoren Klassen zu erstellen.

Hier bin ich auf ein 1 zu n Problem gestoßen in der Sensor Klasse. Da in der Konfiguration alle Sensoren einen Sensor Block aber n Eval Blöcke besitzen. Da ich nicht beurteilen konnte ob und wie viele weitere Sensoren n Eval Blöcke besitzen habe ich mich dafür entschieden diese Informationen in der ExcelDB zu hinterlegen. Dadurch dieses Problem nicht nur für 2 Eval Blöcke wie in meinem Fall bei dem Temperatur- und Feuchtigkeitssensors zu lösen. Dies sehen sie hier in der Sensor Klasse innerhalb der Funktion create\_eval\_section()

1. def create\_eval\_section(self):
2. formula\_splited, statistics = self.convert\_values\_from\_db()
3. for x in range(self.get\_number\_of\_evals()):
4. self.cfg = copy.deepcopy(DEFAULT\_SETTINGS.get(self.cfg["mess\_typ"].split(", ")[x], {})) | copy.deepcopy(self.cfg)
5. self.select\_channel(self.available\_channels)
6. eval\_dict = copy.deepcopy(DEFAULT\_EVAL\_BLOCK)
7. eval\_dict["statistics"] = statistics[x]
8. self.activision\_values\_stat = statistics[x]
9. eval\_dict["label"] = self.name
10. eval\_dict["formula"] = formula\_splited[x]
11. eval\_dict["formula\_params"] = f"{self.channel.split(',')[x]}" + (f",{self.cfg.get('formula\_params').split('/ ')[x]}" if self.cfg.get("formula\_params") is not None else "")
12. eval\_dict["type"] = self.cfg["eval\_type"].split(", ")[x]
13. eval\_dict["unit"] = self.cfg["unit"]
14. self.create\_activision\_values(x)
15. self.process\_eval\_section(eval\_dict, x)
16. self.eval\_dicts.append(eval\_dict)

Diese Funktion stellt sicher das n Eval Blöcke verarbeitet werden können. Die gesamte Klasse SensorBase finden Sie in Anhang A10.

Der Generator erstellt die Sensor Objekte nacheinander wie sie im Herstellungsplan des jeweiligen Systems hinterlegt sind. Dies war wichtig damit die neuen Konfigurationen demselben Schema entsprechen wie die bisherigen, da der Kunde diese Konfiguration ebenfalls einsehen und ggf. bearbeiten können muss, falls sich vor Ort Änderungen ergeben.

Anschließend stellt der Generator seine Konfiguration zusammen und schreibt Sie in eine .txt Datei. Dies passiert in einem Ordner innerhalb des Firmenprojektes. Dies ist auch das übliche Vorgehen im Tagesgeschäft. Dass die Konfiguration, nachdem sie auf dem Datenlogger fertig gestellt wurde einmal vom Datenlogger gezogen wird und im Firmenprojektordner gespeichert wird.

Die Klasse SensorBase findet sich im Anhang [A10](#_bookmark74).

ZUKURZ

**6 Testphase**

Es wurden Tests mit Falschem Input vorgenommen. Es wurde ein Herstellungsplan mit einem anderen Datenlogger als Pfad eingefügt. Für andere Datenlogger als ein Meteo 40 S/M/L darf das Programm nicht funktionieren. Das Ergebnis des Tests finden sie in Anhang A7 Screenshots / Abbildung 8 Ergebnis des Tests (Falscher Input)

Das Programm wurde über GIT auf anderen Firmenrechnern geladen und über die Batch Datei ausgeführt. Die Ergebnisse des Tests waren Positiv.

Es lief ohne Anpassungsprobleme auf den Rechnern.

Die Konfigurationsdatei die das Programm erstellt, wurde erfolgreich auf einen Datenlogger hochgeladen und von diesem erfolgreich verarbeitet.

Dies war das Hauptkriterium welches durch diesen Test erfüllt wurde.

Das Projekt wurde offiziell abgenommen vom Leiter der IT sowie dem Leiter der Werkstatt, in welcher die Systeme gebaut werden. Beide waren sehr zufrieden mit dem Ergebnis. Dies wird der Firma viel Arbeitszeit sowie dadurch verursachte Kosten sparen und das Tagesgeschäft entlasten.

**7 Abnahmephase**

Das Projekt liegt in GitHub und wird darüber auch verteilt in der Firma. Das Projekt wurde in das GitHub der Firma geladen und so zugänglich gemacht. Die Mitarbeiter, die betroffen sind, wurden in dieses Repo eingeladen und sind so fähig dieses Programm auf den eigenen Rechnern laufen zu lassen. Bei Aktualisierungen ist der Admin dazu angehalten eine Rundmail zu schreiben, um die Mitarbeiter darauf hinzuweisen einen Git Pull durchzuführen. Alle nötigen Schritte dafür wurden in der README hinterlegt.

Es wurde eine Halbstündige Einführung durchgeführt bei 2 Mitarbeitern da das Programm sehr einfach zu bedienen ist. wurde nur erwähnt, wie es über die Batch ausgeführt wird und wo der zu übergebende Pfad liegen muss. Sowie was passieren sollte, wenn Zertifikate nicht gefunden werden. Auch dazu wurden alle nötigen Schritte sowie alles zu Beachtende in der README hinterlegt.

Da ich zur Zeit der Admin des Projektes bin, sowie in näherer Zukunft auch bleiben werde, konnte hier in absehbarer Zeit auf Schulungsmaßnahmen verzichtet werden. Sollte diese Aufgabe in Zukunft auf einen anderen Entwickler übergehen so wird dieser Anhand der Doc Strings innerhalb des Projektes sowie einer kurzen Einarbeitungsphase in der Lage sein das Programm zu begreifen sowie zu warten und zu erweitern

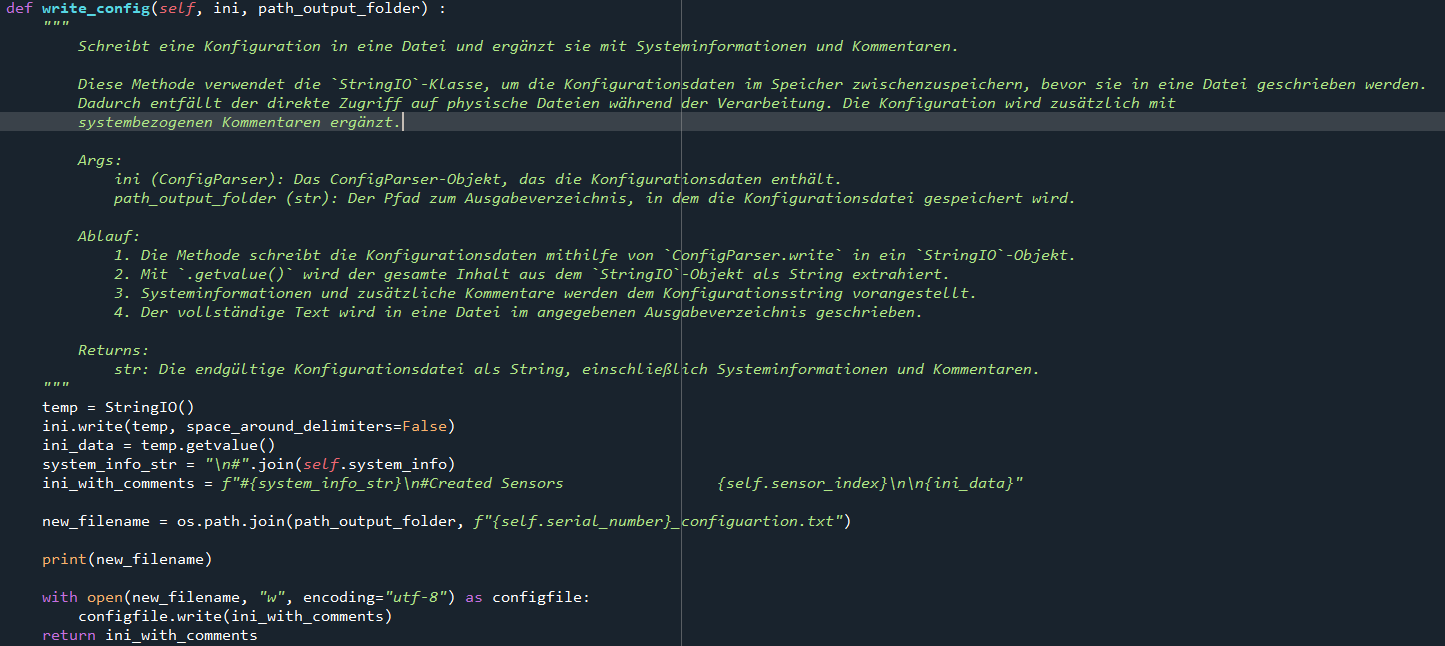
**Beispiel**

Einen Auszug aus der README finden Sie in Anhang A7 Anwender Dokumentation

**8 Dokumentation – Vor Abnahmephase**

* + - 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).
    - In der Entwicklerdokumentation habe ich mich bewusst für ausführliche Doc Strings entschieden. Diese bieten eine präzise Beschreibung der Funktionsweise, der erwarteten Eingaben, der durchgeführten Aktionen und der Rückgabewerte jeder Funktion. Sie sind gut für Einzel Projekte geeignet sowie für kleiner Teams. Zusätzlich habe ich sprechende Variablennamen genutzt.

Ein Beispiel hierfür finden Sie hier:



* Diese Entwicklerdokumentation wurde über die Sphinx Bibliothek in ein HTML-Dokument überführt. Einen Ausschnitt davon finden Sie in Anhang A7 / Entwickler Doku

**Beispiel**

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

**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 [3](#_bookmark46) zu erkennen ist, konnte die Zeitplanung bis auf wenige Ausnahmen eingehal- ten werden.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Fazit** | |  |  |  |  | | --- | --- | --- | --- | | Phase | Soll | Ist | Abweichung | | Analysephase | 8 Stunden | 8 Stunden | 0 Stunden | | Entwurfsphase | 11 Stunden | 10 Stunden | - 1 Stunde | | Entwicklungsphase | 40 Stunden | 41 Stunden | + 1 Stunde | | Testphase | 7 Stunden | 7 Stunden | 0 Stunden | | Dokumentationsphase | 14 Stunden | 14 Stunden | 0 Stunden | | Gesamt | 80 Stunden | 80 Stunden | 0 Stunden | |
|  |
|  | **Tabelle 3: 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, Kim Christopher Plecker, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit 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

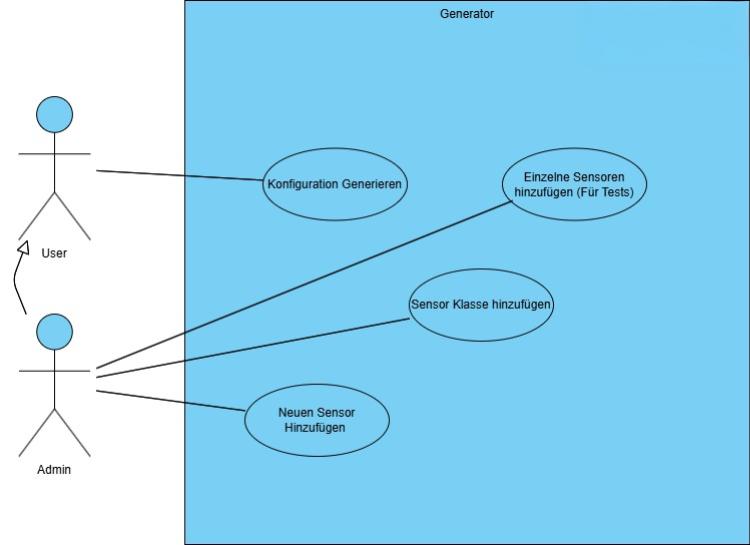
**Anhang**

## A1 Detaillierte Zeitplanung

|  |  |
| --- | --- |
| 1. Analysephase | 8 Stunden |
| 1.1 Erstellung der IST Analyse | 2 Stunden |
| 1.1.1 Fachgespräch mit Verantwortlichen | 1 Stunden |
| 1.1.2 Analyse des bisherigen Prozesses | 1 Stunden |
| 1.2 Ermittlung des Sollzustands | 4 Stunden |
| 1.2.1 Fachgespräch mit Verantwortlichen | 2 Stunden |
| 1.2.2 Zukünftigen Prozess definieren | 2 Stunden |
| 1.3 Wirtschaftlichkeitsanalyse | 2 Stunden |
| 1.3.1 Analyse aller Kosten | 1 Stunden |
| 1.3.2 Make or Buy Entscheidung | 0,5 Stunden |
| 1.3.3 Amortisierungszeitraum berechnen | 0,5 Stunden |
| 1. Entwurfsphase | 11 Stunden |
| 2.1 Datenverarbeitungsentwurf erstellen | 4 Stunden |
| 2.1.1 Daten Analysieren | 1,5 Stunde |
| 2.1.2 Verarbeitungskonzept erstellen | 1,5 Stunde |
| 2.1.3 Grob Klassen entwerfen | 1 Stunde |
| 2.2 Feinere Klassen entwerfen | 2 Stunden |
| 2.2.1 Sensor Klasse definieren | 1 Stunde |
| 2.2.2 Generator Klasse definieren | 1 Stunde |
| 2.3 Klassendiagramm entwerfen | 3 Stunden |
| 2.4 Interface Entscheidung | 2 Stunden |
| 2.4.1 Fachgespräch mit Verantwortlichen | 2 Stunden |
| 1. Entwicklungsphase | 40 Stunden |
| 3.1 Entwicklung der Sensor Klasse / Klassen | 10 Stunden |
| 3.1.1 Analyse der Daten | 4 Stunden |
| 3.1.2 Erstellung der Sensor Klassen | 6 Stunden |
| 3.2 Entwicklung der Datenbank in Excel | 5 Stunden |
| 3.2.1 Analyse der Daten | 2 Stunden |
| 3.2.2 Erstellung der Datenbank | 3 Stunden |
| 3.3 Entwicklung der Generator Klasse | 10 Stunden |
| 3.3.1 Analyse der Daten des Datenloggers | 1 Stunde |
| 3.3.2 Entwicklung des Generators | 8 Stunden |
| 3.3.3 Erstellung der statischen Datei | 1 Stunde |
| 3.4 Entwicklung des Excel Readers | 3 Stunden |
| 3.4.1 Analyse der Möglichkeiten | 1 Stunde |
| 3.4.2 Entwicklung des Excel Readers | 2 Stunden |
| 3.5 Entwicklung des PDF Readers | 5 Stunden |
| 3.5.1 Analyse der Möglichkeiten | 1 Stunde |
| 3.5.2 Analyse welche Daten benötigt werden | 0,5 Stunden |
| 3.5.3 Entwicklung des PDF Readers | 2 Stunden |
| 3.5.4 Entwicklung des PDF Filters | 1,5 Stunden |
| 3.6 Entwicklung des User Interfaces | 1 Stunde |
| 3.6.1 Entwicklung des User Interfaces | 1 Stunde |
| 3.7 Zusammenführung aller Komponenten | 6 Stunden |
| 3.7.1 Daten Importieren | 1 Stunde |
| 3.7.2 Erste Funktionstest | 2 Stunden |
| 3.7.3 Dateien Debuggen | 3 Stunden |
| 1. Testphase | 7 Stunden |
| 4.1 Generierung von Testfällen | 3 Stunden |
| 4.1.1 Festlegen verschiedener Testarten | 1 Stunde |
| 4.1.2 Generierung von Konfigurationen und vergleich mit bestehenden | 2 Stunden |
| 4.2 Software auf Hardwarekompatibilität Testen | 1,5 Stunden |
| 4.2.1 Software auf fremden Rechnern Testen | 0,75 Stunden |
| 4.2.2 Software auf Ubuntu Testen | 0,75 Stunden |
| 4.3 Software mit Fehlerhaftem Input Testen | 1,5 Stunden |
| 4.3.1 Tests durchführen | 1,5 Stunden |
| 4.4 Abgabe des Projektes | 1 Stunde |
| 4.4.1 Kurze Vorstellung sowie Erläuterung | 1 Stunde |
| 1. Dokumentationsphase | 14 Stunden |
| 5.1 Dokumentation innerhalb des Codes | 1 Stunde |
| 5.2 Projektdokumentation erstellen | 12 Stunden |
| 5.3 Doku für die Inbetriebnahme schreiben | 1 Stunde |

**Tabelle 3: Detaillierte Zeitplanung**

## A2 Use-Case-Diagramm

****

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

## A3 Ereignisgesteuerte Prozesskette

**Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung**

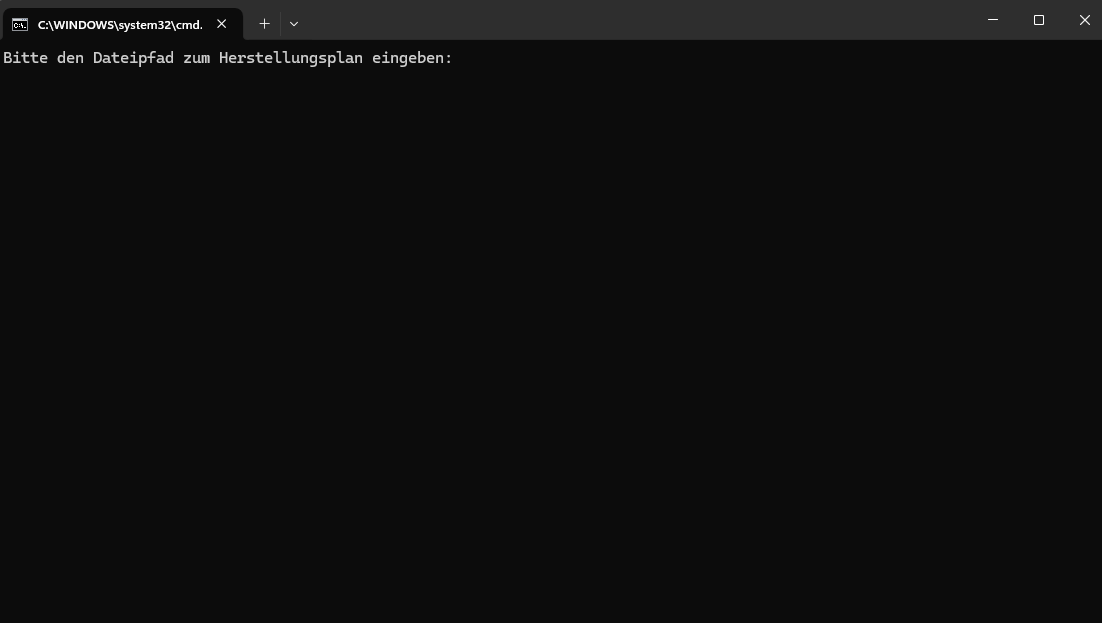
**Nach Pfad ungültig auf selbes Programm Ende – Fehlermeldung – Orange – Ende Grün /**

**Sensor Erstellen muss auch Orange**

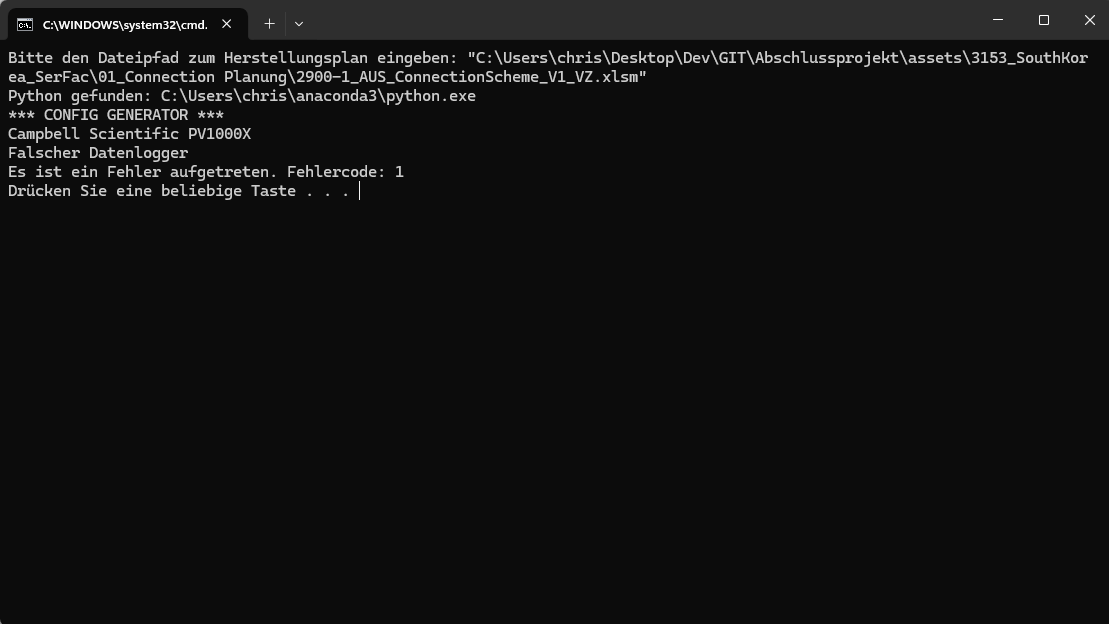
**Ereignisse nach Funktionen / Ereignis – Ergebnis**

**Abbildung 2: EPK des Generators**

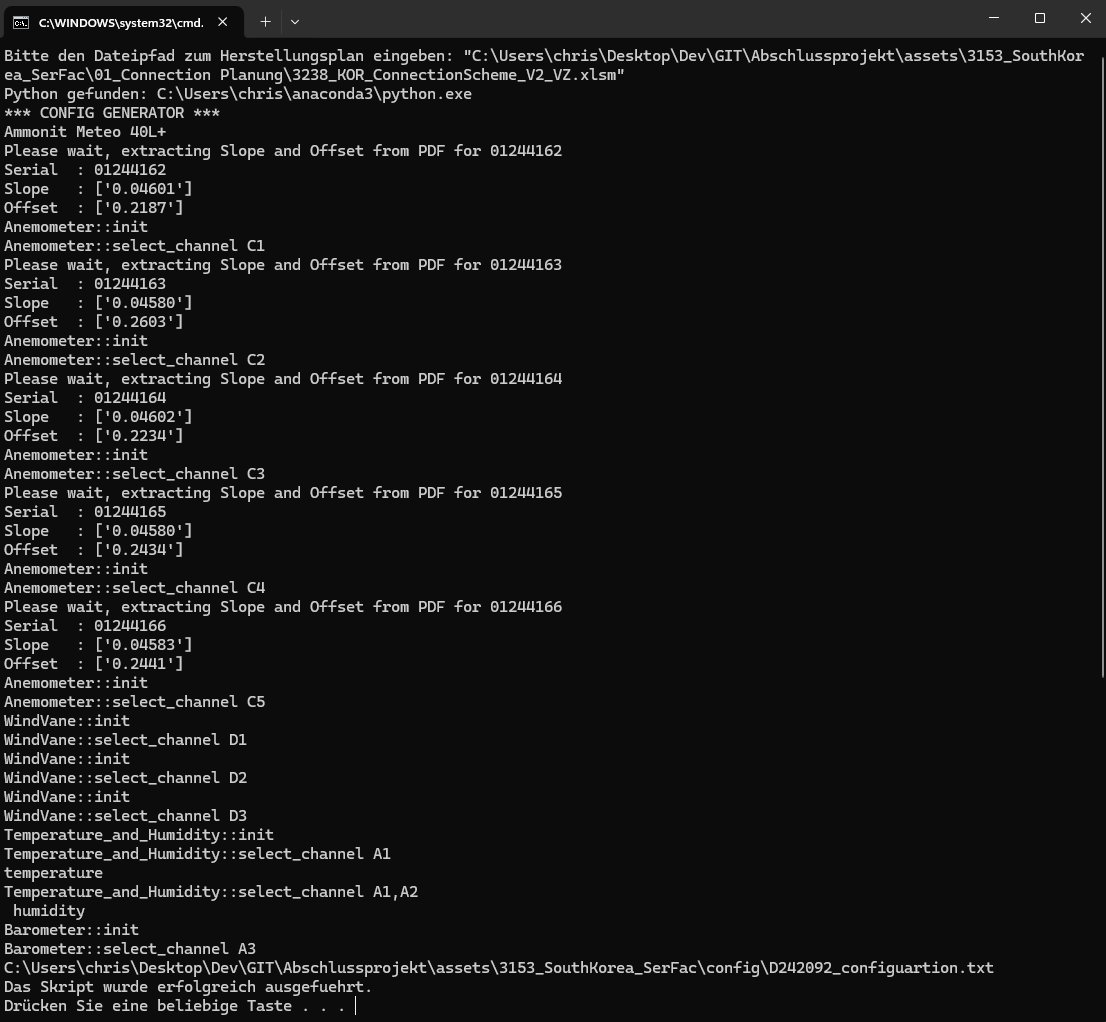
## A4 Screenshots der Anwendung



**Abbildung 3: Aufruf des Programms über Batch Datei**

****

**Abbildung 4: Ergebnis des Tests (Falscher Input)**

****

**Abbildung 5: Erfolgreiche Ausführung des Programms**

**Ein Bild, das Text, Software, Multimedia-Software, Computersymbol enthält.

Automatisch generierte Beschreibung**

**Abbildung 6: Konfiguration auf Ammoniten Aufspielen (Zu Testzecken auf neuer Datenlogger Generation - Ammonit Meteo 42 – Test Erfolgreich)**

## A5 Entwickler Doku

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

Automatisch generierte Beschreibung

**Abbildung 7: Auszug Entwickler Doku**

## A6 Klasse: SensorBase

Kommentare sowie Doku Strings werden nicht gezeigt.

class SensorBase:

    def \_\_init\_\_(self, sensor\_index, eval\_index, available\_channels, SENSOR\_DB, \*\*kwargs):

        print(f"{type(self).\_\_name\_\_}::init")

        self.sensor\_index = sensor\_index

        self.eval\_index = eval\_index

        self.available\_channels = available\_channels

        self.cfg = kwargs

        self.name = self.cfg.get('name',"unnamed")

        self.height = self.cfg.get('height',0.0)

        self.type\_no = self.cfg.get('type\_no',None)

        self.serial = self.cfg.get('serial',"SNxxxyyyy")

        self.type\_no = self.cfg.get('type\_no',"")

        self.mess\_typ = self.cfg.get("mess\_typ",[])

        self.channel = None

        self.eval\_dicts = []

        self.create\_eval\_section()

        self.create\_sensor\_section()

        self.update\_eval()

    def \_set\_if\_exists(self,key, name=None):

        if name is None:

            name = key

        if self.cfg[key]:

            self.\_sensor[name] = self.cfg[key]

    def create\_sensor\_section(self):

        self.\_sensor = copy.deepcopy(DEFAULT\_SENSOR\_BLOCK)

        self.\_sensor["model"] = self.cfg["model"]

        self.\_sensor["label"] = self.name

        self.\_sensor["height"] = self.height

        self.\_sensor["type"] = self.cfg["type"]

        self.\_sensor["used\_channels"] = self.channel

        self.\_sensor["evals"] = self.eval\_index

        self.\_sensor["serial\_number"] = self.serial

        self.\_set\_if\_exists("url", name="url\_path")

        self.\_set\_if\_exists("protocol")

        self.\_set\_if\_exists("adc\_range\_u")

        self.\_set\_if\_exists("sensor\_type")

        self.\_sensor["model\_id"] = int(self.cfg["model\_id"])

        self.\_sensor["evals"] = ",".join(str(self.eval\_index + i) for i in range(self.get\_number\_of\_evals()))

        self.process\_sensor\_section()

        result = {f"Sensor\_{self.sensor\_index}" : self.\_sensor}

        return result

    def create\_eval\_section(self):

        formula\_splited, statistics = self.convert\_values\_from\_db()

        for x in range(self.get\_number\_of\_evals()):

            self.cfg = copy.deepcopy(DEFAULT\_SETTINGS.get(self.cfg["mess\_typ"].split(", ")[x], {})) | copy.deepcopy(self.cfg)

            self.select\_channel(self.available\_channels)

            eval\_dict = copy.deepcopy(DEFAULT\_EVAL\_BLOCK)

            eval\_dict["statistics"] = statistics[x]

            self.activision\_values\_stat = statistics[x]

            eval\_dict["label"] = self.name

            eval\_dict["formula"] = formula\_splited[x]

            eval\_dict["formula\_params"] = f"{self.channel.split(',')[x]}" + (f",{self.cfg.get('formula\_params').split('/ ')[x]}" if self.cfg.get("formula\_params") is not None else "")

            eval\_dict["type"] = self.cfg["eval\_type"].split(", ")[x]

            eval\_dict["unit"] = self.cfg["unit"]

            self.create\_activision\_values(x)

            self.process\_eval\_section(eval\_dict, x)

            self.eval\_dicts.append(eval\_dict)

    def convert\_values\_from\_db(self):

        if "," in self.cfg["formula"]:

            formula\_splited = self.cfg["formula"].split(", ")

        elif "," not in self.cfg["formula"] and self.get\_number\_of\_evals() > 1 :

            formula\_splited = [self.cfg["formula"]] \* self.get\_number\_of\_evals()

        else:

            formula\_splited = [self.cfg["formula"]]

        if "/" in self.cfg["statistics"] :

            statistics = self.cfg["statistics"].split("/")

        if not "/"  in self.cfg["statistics"] :

            statistics = []

            statistics.extend([self.cfg["statistics"]] \* self.get\_number\_of\_evals())

        return formula\_splited, statistics

    def update\_eval(self):

        for x in range(self.get\_number\_of\_evals()):

            self.eval\_dicts[x]["sensors"] = self.sensor\_index

    def create\_activision\_values(self, x):

        self.activision\_values = {"active": True}

        if "A" in self.channel:

            self.process\_activision\_values(x)

        else:

            self.process\_activision\_values(x)

    def process\_activision\_values(self, x):

        pass

    def get\_number\_of\_evals(self):

        return int(self.cfg["counter\_evals"])

    def select\_channel(self, channels):

        channel\_type = self.cfg["channel\_type"]

        channel\_for\_sensor = channels[channel\_type].pop(0)

        if not self.channel:

            self.channel = f"{channel\_type}{channel\_for\_sensor}"

        else:

            self.channel += "," +  f"{channel\_type}{channel\_for\_sensor}"

        print(f"{type(self).\_\_name\_\_}::select\_channel {self.channel}")

        return self.channel

    def process\_sensor\_section(self):

        pass

    def process\_eval\_section(self):

        pass

    def get\_sections(self):

        s = {f"Sensor\_{self.sensor\_index}": self.\_sensor}

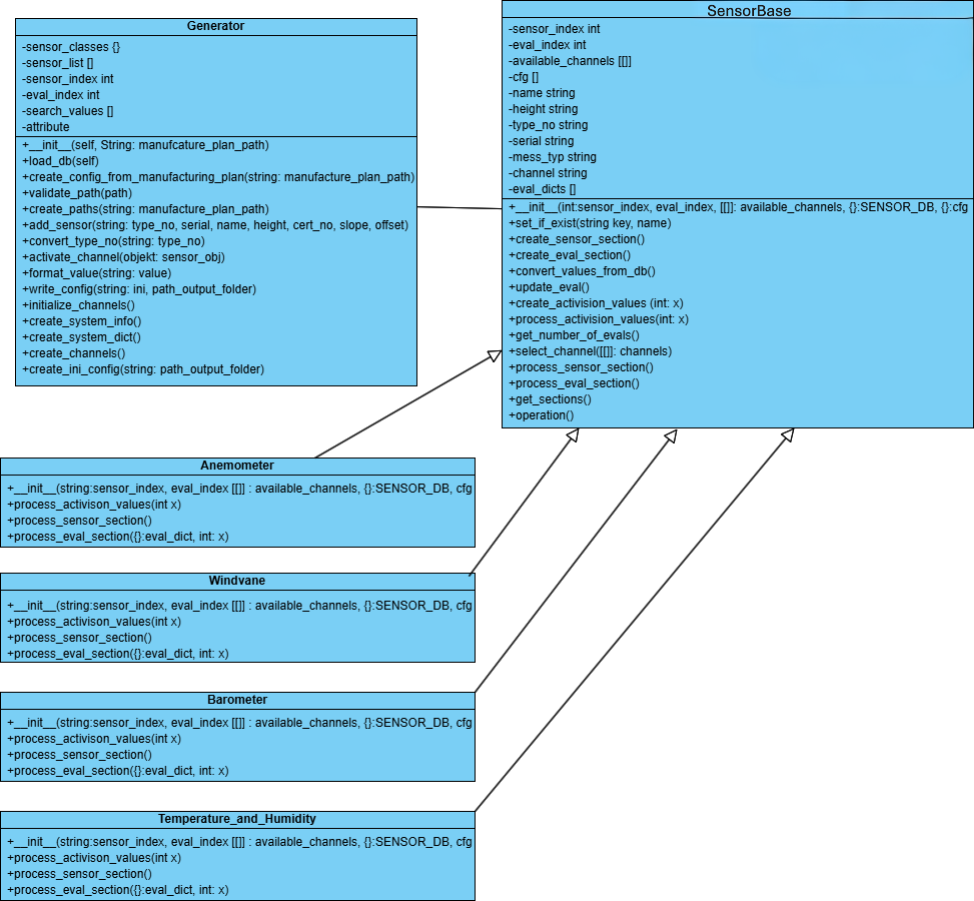
        evals = {f"Eval\_{self.eval\_index + index}": eval\_dict for index, eval\_dict in enumerate(self.eval\_dicts)}

        sections = [{k: v} for k, v in s.items()]

        sections.extend([{k: v} for k, v in evals.items()])

        return sections

## A7 Klassendiagramm

****

**Abbildung 8: Klassendiagramm**