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

Prüflingsnummer 50089

Ausbildungsbetrieb

BFW Würzburg

Helen-Keller-
Straße 5

97209 Würzburg

Praktikumsbetrieb:

ProfEC Ventus

Am Patentbusch 3B

26125 Oldenburg

Inhaltsverzeichnis

Inhaltsverzeichnis

Tabellenverzeichnis

Glossar

BAT ..... *Batch*
CMD ..... *Command Prompt*
EPK ..... *Ereignisgesteuerte Prozesskette*
GUI ..... *Graphical User Interface*
HTML ..... *Hypertext Markup Language*
LAN ..... *Local Area Network*
SQL ..... *Structured Query Language*

1. Einleitung

In diesem Kapitel stelle ich die Firma, das Projektumfeld und das Projekt vor. Beim Projekt handelt es sich um die Entwicklung eines Generators für Konfigurationsdateien für einen Datenlogger, der im Unternehmen eingesetzt wird.

1.1 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.

1.2 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.

1.3 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.

1.4 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.

1.5 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

In diesem Kapitel stelle ich die Projektphasen sowie die Projektplanung vor. Punkte wie die Projektphasen, Zeitplanung und Abweichungen vom Projektantrag werden erläutert.

2.1 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.

Eine detailliertere Zeitplanung ist in Tabelle 3 in **Anhang A1** 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

Tabelle 1 zeigt ein die grobe Zeitplanung meines Projektes

2.2 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

2.3 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

2.4 Entwicklungsprozess

Der Entwicklungsprozess wird dem Klassischem Wasserfallmodell folgen.

Die Phasen des Wasserfallmodells:

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

3. Analysephase

In diesem Kapitel werde ich die grundlegenden Analysen durchführen und erläutern. Unter anderem werden hier eine Ist-Analyse und eine Wirtschaftlichkeitsanalyse durchgeführt. Zusätzlich werden Anwendungsfälle sowie Qualitätsanforderungen beschrieben.

3.1 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.

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.

3.2 Wirtschaftlichkeitsanalyse

Auflistung der Fixkosten:

1. Arbeitszeitkosten Azubi: 0€ (Unbezahltes Praktikum)
2. Büroraum (inkl. Nebenkosten, Reinigung, etc.): 24,83€ pro Tag (Quelle Der große Preisvergleich: Coworking vs. Büro? | SleevesUp!)
3. Zeitkosten des beratenden Mitarbeiters: 10 Stunden a 100€ = 1000€
4. Kostenpflichtige Software: Microsoft 365 Business Standard: 11,70€/Monat
5. Schulung und Einführung für Mitarbeiter: 3 Stunden a 40€
6. Abnahme durch Fachabteilung: 1 Stunde a 80€
   Gesamt: 1.763€

Auflistung der laufenden Kosten:

Einfügen eines bereits als Klasse angelegten Sensors in das Programm:

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

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

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

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

1. Fixkosten: 1.763€
2. Absehbare laufende Kosten pro Jahr: 1.440€
3. Gesamtkosten im Zeitraum von Jahr 1: 3.203€
4. 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.**

3.2.1 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.

3.2.2 Amortisationsdauer

Gesamtkosten Jahr 1 : 3.203€

Gesamtersparnis pro Jahr : 10.400€

Gesamtkosten durch Gesamtersparnis:

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 gleichbleiben.

3.3 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:

1. Der User bekommt eine Schnittstelle, über welcher er das Programm ausführen kann sowie einen Pfad übergeben kann.
2. Der Admin bekommt ebenfalls die Möglichkeit das Programm auszuführen sowie das System zu erweitern, zu pflegen, Fehler zu beheben.

Ein Use-Case-Diagramm finden Sie im **Anhang A2**.

3.4 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

In diesem Kapitel wird die Entwurfsphase des Projekts beschrieben. Dabei werden unter anderem die Zielplattform, die Architektur, die Benutzeroberfläche, das Datenverarbeitungskonzept sowie Maßnahmen zur Qualitätssicherung erläutert.

4.1 Zielplattform

Das Projekt wird in Python umgesetzt und Excel als Datenbankersatz genutzt.

Die Software wird auf allen Windows Rechnern und Linux Rechnern welche

Python fähig sind, beziehungsweise auf denen Python standartmäßig installiert ist, einsetzbar sein.

Die Kriterien dafür sind:

Einfache Bedienbarkeit: Das Programm soll auch von projektfremden Personen problemlos ausführbar sein.

Wartbarkeit: Der Code muss für andere Entwickler leicht verständlich und wartbar sein.

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.

4.2 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 und die zukünftige Erweiterung des Programms einzelne Klassen und Ihre Funktionen können leicht wiederverwertet werden.

Folgende Programmierparadigmen sollen berücksichtigt werden:

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

4.3 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 ausführen der Batch Datei, finden Sie im **Anhang A4**.

4.4 Datenmodell

Es werden folgende Klassen verwendet

Sensoren:

SensorBase, Anemometer, Barometer, Windvane, Temperature_and_Humidity.

Generator:

Generator.

Zusätzlich werden folgende Hilfsfunktionen und Daten erstellt:

Ein Dictionary welches Statische Informationen bereit hält

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 A4** / Aufspielen der Konfiguration

4.5 Geschäftslogik

Es gibt 2 Möglichkeiten das Programm auszuführen:

Der User wird innerhalb einer Entwicklungsumgebung (z.B. Spyder - Spyder wird hier explizit genannt, da es die gängige Umgebung innerhalb des Unternehmens 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.

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 und 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 A7** eingesehen werden.

Die EPK in **Anhang A3** zeigt den grundsätzlichen Ablauf des Generators.

4.6 Maßnahmen zur Qualitätssicherung

Es wurden folgende Testfälle festgelegt:

Generierung und Vergleich von bestehenden Konfigurationen.

Konfigurationen auf den Datenlogger laden und Testen ob der Datenlogger diese verarbeiten kann.

Ein Test mit den Anwendern, um zu gewährleisten, dass eine Fehlbedienung ausgeschlossen ist.

Die Installation und Ausführung des Programms erfolgt auf firmeninternen Rechnern, um mögliche Komplikationen auszuschließen.

4.7 Datenverarbeitungskonzept

Sensoren Klassen:

Anemometer, Barometer, Temperature_and_Humidity, Windfahne, Barometer

Generator Klasse:

Generator

Unterstützende Funktionen:

PDF Extractor, Herstellungsplanleser, Daten Dictionary

1. Der User erzeugt eine Instanz der Generator Klasse und übergibt seinen Dateipfad im Konstruktor oder in der Shell.
2. Der Generator liest den Herstellungsplan ein und 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, 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.
6. Nun kann der User im Stammverzeichnis des Projektes die INI Datei in den Datenlogger hochladen.

Der Generator übernimmt in meinem Projekt die zentrale Rolle als datenverarbeitende Klasse. Er extrahiert die relevanten Daten aus der Excel-Datei, indem er den vom Benutzer übergebenen Pfad nutzt, und greift dabei auf eine externe Hilfsfunktion zu. Anschließend erstellt der Generator eigenständig Sensorobjekte und nutzt den PDF-Extractor, um die Zertifikate auszuwerten. Die erstellten Sensorobjekte fügt er seiner internen Datenstruktur hinzu, verarbeitet diese nacheinander und wertet sie aus. Sobald alle Sensorobjekte vollständig verarbeitet sind, generiert der Generator eine Konfigurationsdatei im INI-Format und speichert diese als Textdatei.

5. Implementierungsphase

In dieser Projektphase erläutere ich wie ich die Datenstrukturen und die Benutzeroberfläche erstellt habe. Sowie einige interessante Probleme sowie deren Lösungen auf die ich während der Entwicklung gestoßen bin.

5.1 Implementierung der Datenstrukturen

Ich habe bestehende Konfigurationen für den Datenlogger miteinander verglichen und daraus Werte extrahiert die in allen, mir bekannten, Sensoren vorkommen ausgewertet. Ich habe diese Werte als Standard für alle zu konfigurierenden Sensoren festgelegt. Die in der Excel-Datenbank gespeicherten Daten enthalten teilweise diese Standardwerte sowie zusätzliche Informationen, die ich während der Programmierung entdeckt habe. Die Informationen wiederholen sich teilweise. In einer SQL-Datenbank würde dies in mehrere Tabellen aufgeteilt werden. Darauf wurde hier verzichtet, da in der Firma bisher keine SQL-Datenbankstrukturen eingesetzt werden, das Fachwissen, diese Datenbanken zu verwalten und zu benutzen bisher fehlt. Ich habe mich für einen Objektorientieren Ansatz entschieden da dieser es ermöglicht das Programm leicht zu erweitern, zu warten und zu debuggen. Der Generator ist sozusagen die Masterklasse, die den Herstellungsplan in der Hand hält und daraus dynamisch die Sensor Objekte instanziiert und diese Objekte in sich Speichert.

5.2 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. Zusätzlich ist möglich den Generator über eine IDE auszuführen. Wenn das Skript geöffnet wird, gibt es unten die Main in welcher eine Variable (Filepath) erwartet wird. Sobald dieser eingegeben wird, kann auf Play gedrückt werden und der Konfigurator erstellt die Konfiguration.

5.3 Implementierung der Geschäftslogik

Als erstes stellte sich die Frage, wo und wie die notwendigen Daten für die jeweiligen Sensoren und für den Generator hinterlegt werden. Dies führte zu einer Excel Datenbank für Sensorwerte sowie einem statischen Dictionary 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()

```
        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(',')[
    + (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)
```

Abbildung 1 create_eval_section()

Diese Funktion stellt sicher das n Eval Blöcke verarbeitet werden können.

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.

Beim Generator bin ich während der Entwicklung auf die Frage gestoßen: Wie sollen die Sensoren dem Generator zugefügt werden und wie soll die Konfiguration erstellt werden? Ich habe dieses Problem über eine Ableitung des Builder Pattern gelöst. Da meine Konfiguration aus vielen kleineren, aber komplexen Objekten sowie Informationen besteht, war diese abgeleitete Lösung die Lösung die eingesetzt wurde. Die Funktion

create_config_from_manufacturing_plan(self, manufacturing_plan_path) ist die Ableitung aus dem Builder Pattern. Sie erwartet den Herstellungspfad. Dieser wird Validiert durch den Aufruf einer validate_path() Funktion. Ist er gülitg werden die Pfade des Output Ordners sowie des Zertifikats Ordners durch die Funktion Create_Paths() erstellt und zurück gegeben. Danach wird die Excel Datei ausgelesen und ein Dataframe und eine Liste mit Projektinformationen zurück gegeben über die Funktion excel_reader_manufactureplan.read_man_plan(). Die daraus resultierenden Daten in der Liste werden dem Generator zugeordnet. Danach werden die Kanäle initialisiert sowie überprüft ob im Herstellungsplan der Richtige Datenlogger hinterlegt ist, durch die Funktion initialize_channels(). Ist alles Korrekt geht der Prozess weiter und es werden 3 weitere Funktionen ausgeführt: self.create_system_info(, self.create_channels() self.create_system_dict(). Diese erstellen eine System Information, Die Channels und ein System Dictionary. Nun werden Variablen initialisiert und der Dataframe mit Zwei Schleifen durchlaufen eine für die Zeilen und eine für die Spalten. Hier werden die entsprechenden Werte den Variablen zugeordnet. Sollte eine Zertifikatsnummer gefunden werden wird im Zertifikatsordner das entsprechende Zertifikat ausgelesen (Nur das Jeweilige Zertifikat des passenden Sensors. Dies war wichtig für die Leistungsfähigkeit des Programms) und die Variablen Slope und Offset zugeordnet. Wenn alle Spalten der jeweiligen Zeile durchlaufen worden wird ein Sensor erstellt über die Methode add_sensor() in welcher Slope und Offset standartmäßig mit None erwartet werden. Da nicht jeder Sensor kalibriert ist und diese Werte besitzt. Es gibt Sensoren, die für diesen Fall Standard Werte bereitstellen. Diese wurden in der ExcelDB hinterlegt und später im Programm an den Entsprechenden Stellen eingesetzt. Sobald beide Schleifen durchlaufen worden wird die Funktion create_ini_config() ausgeführt die die Fertige Konfiguration erstellt und zurück gibt. Das Ergebnis dieser Funktion wird anschließend zurückgegeben.

Die Klasse SensorBase findet sich im **Anhang A6**.

Die Funktion create_config_from_manufacturing_plan() finden sie in **Anhang A4** Screenshots / create_config_from_manufacturing_plan

6. Testphase

In dieser Projektphase beschreibe ich welche Tests ich während der Testphase durchgeführt habe. Ich gebe auch zu den Tests ein kurzes Statement ab, ob die Tests erfolgreich verliefen.

6.1 Vergleich bestehender Konfigurationen

Es wurden mehrere Konfigurationen für bereits beim Kunden in Betrieb erzeugt und mit den Originalen auf den Datenloggern laufenden Konfigurationen verglichen. Hier wurde ausschließlich auf die Sensor- und Evalblöcke geachtet und diese miteinander verglichen. Dies wurde gemacht um die Korrekte Implementierung der Sensoren sowie der Slope und Offset Werte zu validieren. Die Tests wurden manuell vorgenommen und waren erfolgreich.

6.2 Validierung der Datenlogger-Kompatibilität

Die im ersten Test erstellten Konfigurationen wurden auf den Datenlogger geladen, um zu sehen, ob der Datenlogger diese verarbeiten kann und ob die Integration der Sensoren reibungslos funktioniert. Der Test verlief erfolgreich.

6.3 Benutzerakzeptanztest und Fehlbedienungssicherheit

Als nächstes wurde die Software einem Mitarbeiter aus der Werkstatt zu Verfügung gestellt, der bisher die Konfiguration Manuell erstellt hat. Dieser wurde dazu angehalten das Programm bei dem neusten Firmen Projekt einzusetzen, welches einen Ammonit Datenlogger nutzt. Dies Tat der Mitarbeiter und war begeistert von der neuen Möglichkeit. Er war sehr zufrieden mit dem Ergebnis sowie der unkomplizierten Methode das Programm über eine Batch Datei zu starten und auszuführen. Die Konfiguration des Produktivsystem wurde aus Gründen der Sicherheit nochmals vom Werkstattleiter überprüft. Auch hier wurden keine Beanstandungen gefunden. Das System wurde dem Kunden ausgeliefert. Da dies der erste Einsatz auf einem Produktivsystem war wird dieses System nun beobachtet, um Komplikationen auf langer Strecke zu erkennen oder auszuschließen. Bisher hat der Kunde das System noch nicht in Benutzung. Daher kann hier noch kein Abschließendes Urteil gefällt werden.

6.4 Interne Installations- und Ausführungstests

Das Programm wurde auf einem Firmenrechner über GIT geladen und die Ausführbarkeit überprüft. Der Test verlief ohne Komplikationen.

7. Dokumentation

In diesem Projektabschnitt erläutere ich für welche Personengruppen, in welcher Tiefe ich Dokumentationen angefertigt habe.

7.1 Entwickler Dokumentation

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 Einzelprojekte geeignet und für kleine Teams. Zusätzlich habe ich sprechende Variablennamen genutzt. Hierbei wurde darauf geachtet die Richtlinien nach PEB 257 einzuhalten.

Ein Beispiel hierfür finden Sie hier:

```
def convert_type_no(self, type_no):
    """
    Konvertiert eine `type_no` in das Format `4.3351.x0.000`.

    Diese Methode nimmt eine `type_no` im Format `4.3351.10.000` oder `4.3351.00.000`,
    teilt sie anhand des Punkts und ersetzt den dritten Teil der `type_no` durch `x0`,
    wenn die Länge der Teile 4 beträgt und der erste Teil `4` ist.

    Args:
        type_no (str): Die zu konvertierende `type_no` im Format `4.3351.x0.000`.
```

```
    Returns:
        str: Die konvertierte `type_no` im Format `4.3351.x0.000`.
    """
```

Aus diesen Doc Strings wurde mittels der Sphinx Bibliothek eine Entwicklerdokumentation in HTML erstellt.

Einen Ausschnitt daraus finden Sie in **Anhang A5**

7.2 Anwender Dokumentation

Die Anwender Dokumentation wurde in das Projektverzeichnis integriert und befindet sich in der README Datei. In dieser ist die Ausführung des Programms beschrieben sowie was das Programm macht. Zusätzlich sind Eventualitäten hinterlegt und deren Auswirkungen und Behebungen von Problemen.

8. Abnahmephase

In diesem Kapitel beschreibe ich wie ich die Software in der Firma verteilt wird, wie und welche Schulungen ich durchgeführt habe. Es wird weiterhin beschrieben wie die Abnahme durchgeführt wurde und ob diese Erfolgreich war.

8.1 Verteilung innerhalb der Firma

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.

8.2 Schulung der Mitarbeiter

Es wurde eine Halbstündige Einführung durchgeführt bei 2 Mitarbeitern da das Programm sehr einfach zu bedienen ist. Hier wurde erwähnt, wie es über die Batch Datei ausgeführt wird und wo der zu übergebende Pfad liegen muss.

Desweitern habe ich erläutert, was passieren sollte, wenn Zertifikate nicht gefunden werden. Dazu wurden alle nötigen Schritte und zu beachtende Punkte in der README hinterlegt.

8.3 Schulung der Entwickler

Es wurde eine 2-stündige Detailliertere Schulung durchgeführt. In dieser wurde das genaue Vorgehen des Generators, die Erstellung der Sensoren, das Auslesen des Herstellungsplans, das Filtern der PDFs und die Vererbung der Konkreten Sensor Klassen aus der Klasse SensorBase erklärt. Dies war notwendig, damit ein anderer Entwickler die Administration des Programms übernehmen kann, falls der Haupt Admin, in diesem Fall Ich, ausfällt oder anderweitig verhindert ist.

8.4 Abnahme

Das Projekt wurde erfolgreich der Geschäftsleitung, dem IT-Vorgesetzten und der Werkstatt präsentiert und offiziell abgenommen. Die einzige erteilte Auflage besteht darin, dass die ersten Systeme, die mit den durch das Programm generierten Konfigurationen das Haus verlassen, sorgfältig überwacht werden. Zusätzlich sollen die Konfigurationen vor dem Versand der Systeme auf dem Datenlogger nochmals auf ihre Korrektheit überprüft werden.

9. Fazit

In diesem Kapitel wird das Projekt abschließend bewertet. Es wird darauf eingegangen, ob die Zeitplanung eingehalten werden konnte, welche neuen Erkenntnisse während der Umsetzung gewonnen wurden und welche Erfahrungen aus dem Projekt mitgenommen werden. Zudem wird ein Ausblick darauf gegeben, wie das Projekt in der Zukunft genutzt und weiterentwickelt werden kann.

9.1 Soll-/Ist-Vergleich bei der Zeitplanung

Wie Sie in Tabelle 3 erkennen können, wurde die Zeitplanung bis auf wenige Ausnahmen eingehalten werden.

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 2: Soll-/Ist-Vergleich

9.2 Projektziel

Das Projektziel die Schritte 3-5 aus Punkt 3.1 Ist Analyse zu Automatisieren wurde erfolgreich implementiert. Somit wurde das Projektziel erreicht.

9.3 Zufriedenheit des Auftraggebers sowie der Stakeholder

Der Auftraggeber und die Stakeholder des Projektes sind mit dem Ergebnis zufrieden. Sobald sich das Programm auf lange Sicht als Fehlerfrei erweist wird sich die Zufriedenheit weiter steigern.

9.4 Lessons Learned

Während des Projekts konnte ich wertvolle Erfahrungen sammeln. Dazu gehören unter anderem:

1. Klassenorientierte Programmierung in Python: Ich habe ein besseres Verständnis für objektorientierte Prinzipien und deren Anwendung in Python entwickelt.
2. Umgang mit Datenstrukturen: Ich habe intensiv mit Datenstrukturen wie Dictionary, Dataframes, Listen und Arrays gearbeitet und deren Einsatzmöglichkeiten besser verstanden.
3. Dateiverwaltung: Ich konnte lernen, wie man in Python Dateien effizient einliest und schreibt.
4. Bibliotheken und Imports: Der korrekte Umgang mit externen Bibliotheken und deren Imports war ein wichtiger Lernaspekt.
5. Projektfokus und Zeitplanung: Durch eine strukturierte Herangehensweise und klare Zeitplanung konnte ich mich auf das Projekt konzentrieren und es außerhalb des Tagesgeschäfts effizient bearbeiten.

Insgesamt hat das Projekt meinen Arbeitsprozess verbessert, da ich lernte, meine Gedanken zu fokussieren und meine Aufgaben besser zu ordnen.

9.5 Ausblick

Das Programm wird in Zukunft kontinuierlich optimiert und erweitert. Es wird in einem Folgeprojekt zum Einsatz kommen, dessen Ziel es ist, eine interne Teststation für Sensoren zu entwickeln. Darüber hinaus gibt es bereits einige Visionen für die weitere Verwendung des Programms, die jedoch noch nicht konkret geplant sind. Eine mögliche Erweiterung wäre beispielsweise, das Programm über eine Web-Oberfläche für unsere Kunden zugänglich zu machen. Dies würde den Kunden ermöglichen, die Funktionen des Programms bequem aus der Ferne zu nutzen und in Echtzeit mit den Ergebnissen zu interagieren, da immer die Möglichkeit besteht das es vor Ort noch zu Änderungen an der System Konfiguration kommt (beispielsweise ein Defekter Sensor, der ausgetauscht wird oder eine Erweiterung des Systems durch weitere Sensoren).

9.6 Fazit

Abschließend kann ich sagen, dass ich in diesem Projekt viele neue und spannende Aufgaben bearbeiten konnte. Ich habe einen tiefen Einblick in die Programmierung mit Python sowie in die Projektdurchführung erhalten. Mein Skillset hat sich deutlich verbessert, insbesondere bei der Namensgebung von Variablen und der Strukturierung des Codes. Auch meine Fähigkeiten in der Zeitplanung und der Fokussierung auf bestimmte Aufgaben haben sich weiterentwickelt.

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

KIM CHRISTOPHER PLECKER

Anhang

Erläuterung Doku Strings werden in den Bildern nicht Angezeigt.

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
2. 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
3. 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
4. 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	1,5 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
Dokumentationsphase	14 Stunden
5. 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

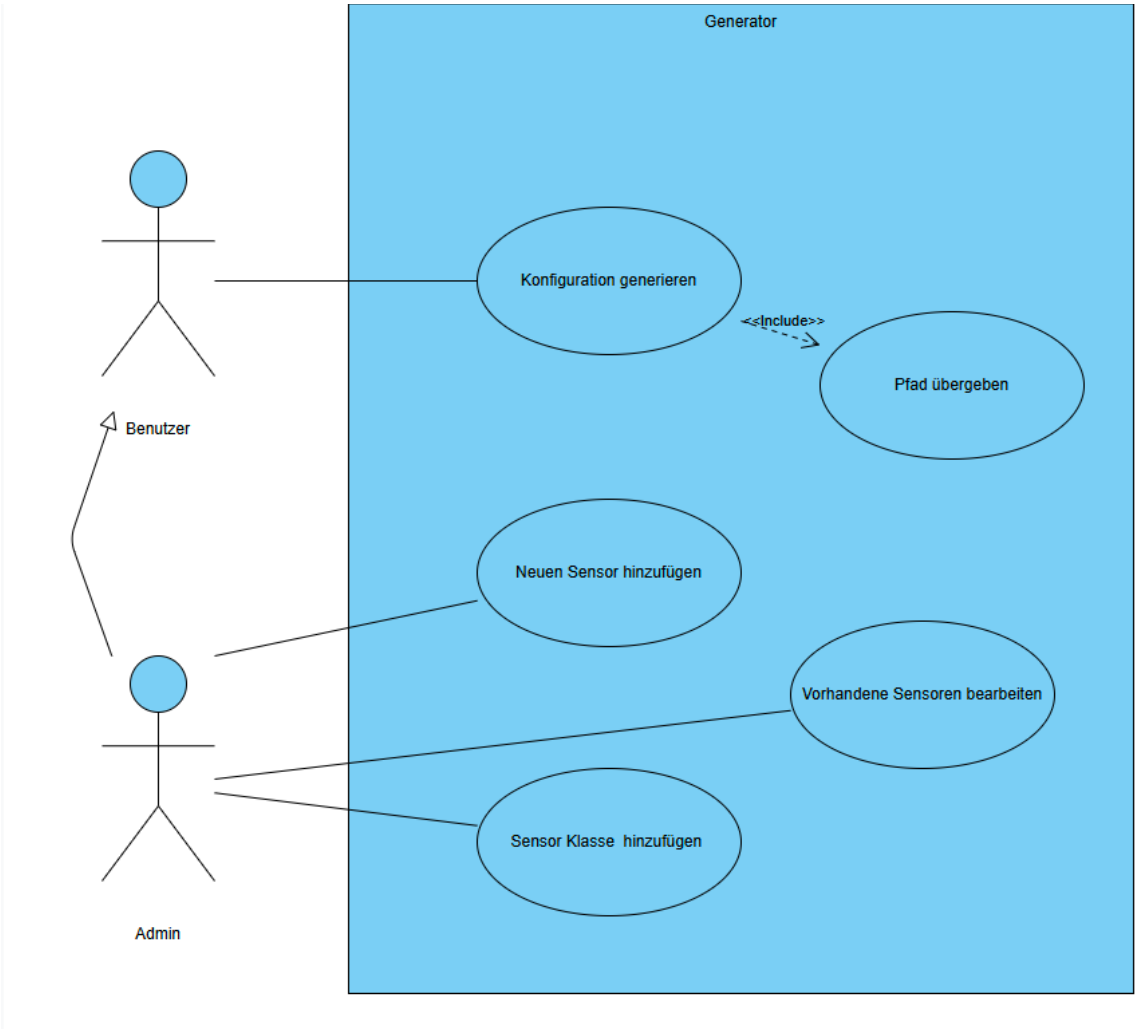

Abbildung 2: Use-Case-Diagramm

A3 Ereignisgesteuerte Prozesskette

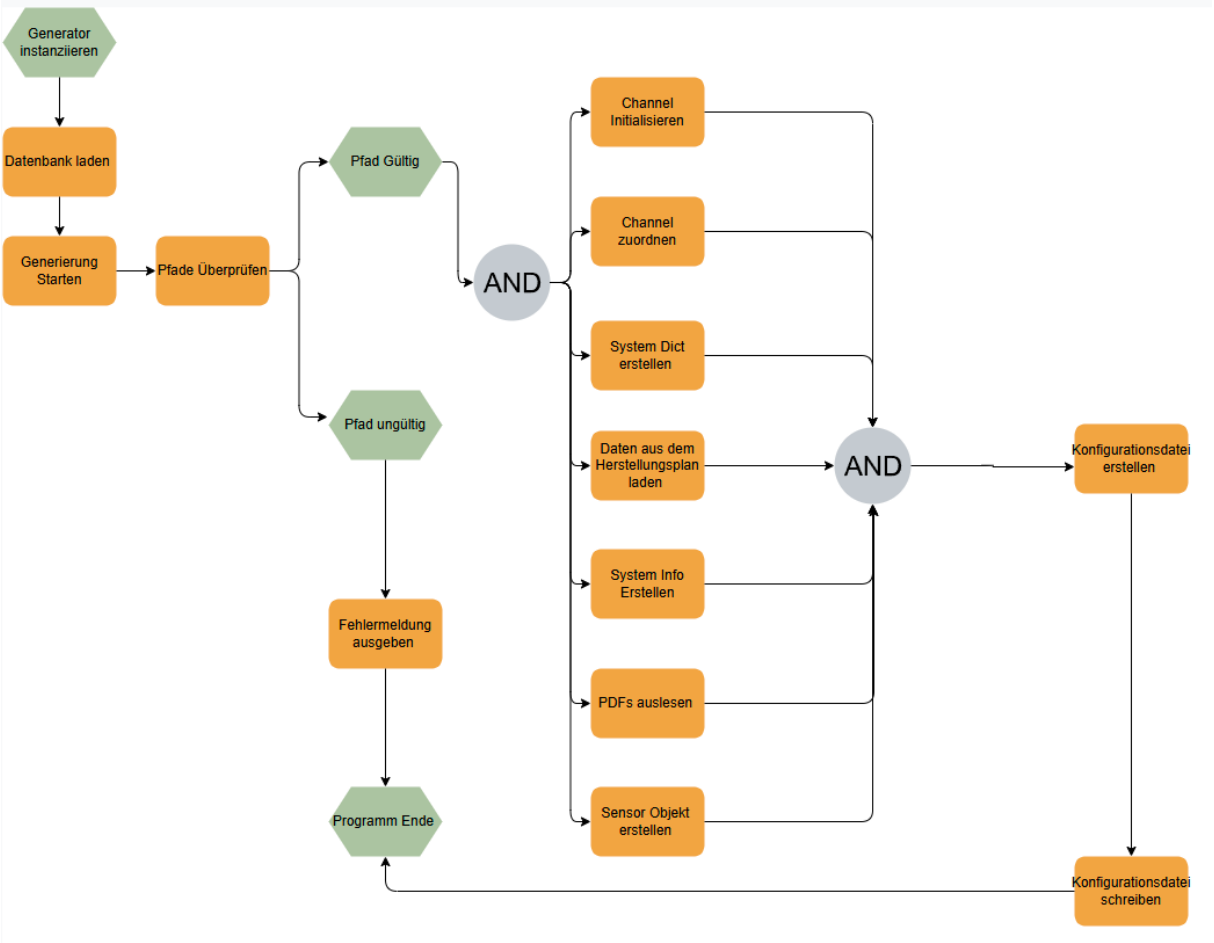

Abbildung 3: EPK des Generators

A4 Screenshots der Anwendung

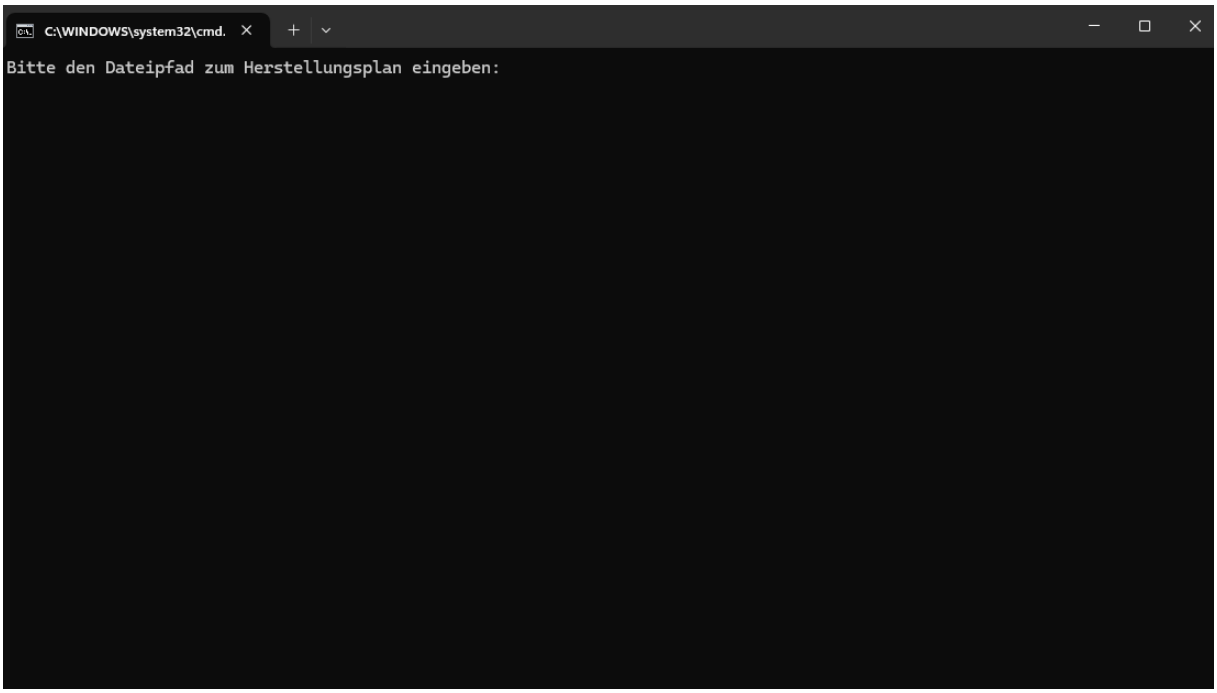

Abbildung 4: Aufruf des Programms über Batch Datei

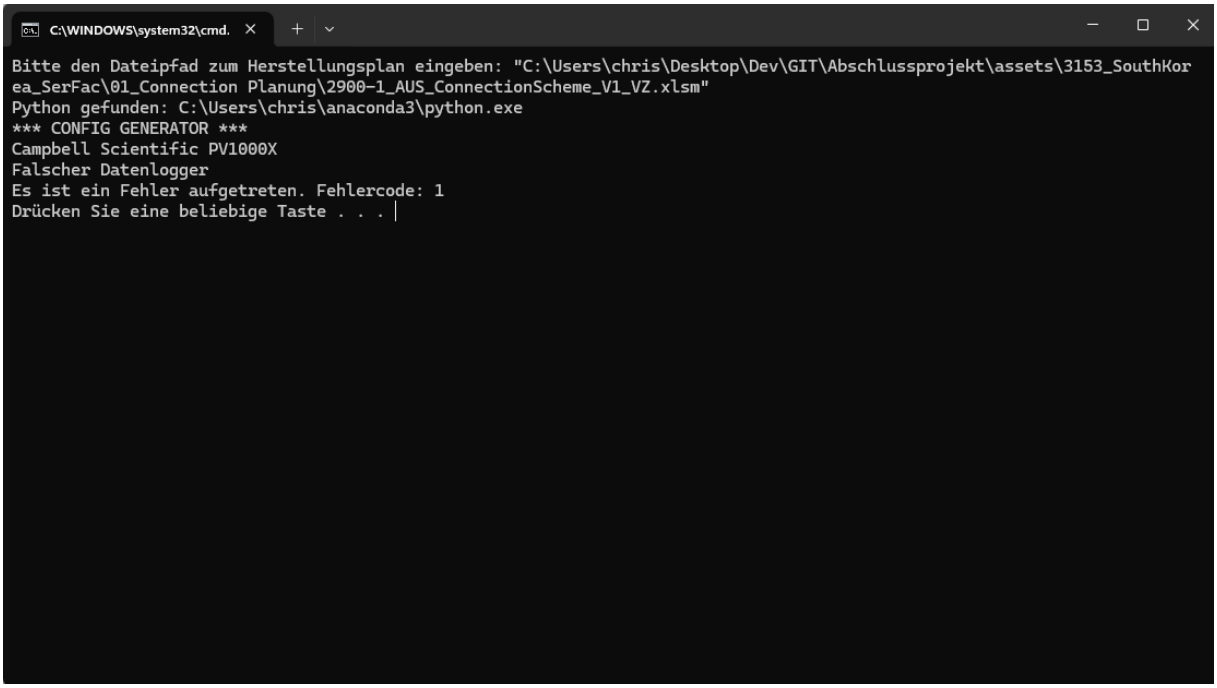

Abbildung 5: Ergebnis des Tests (Falscher Input)

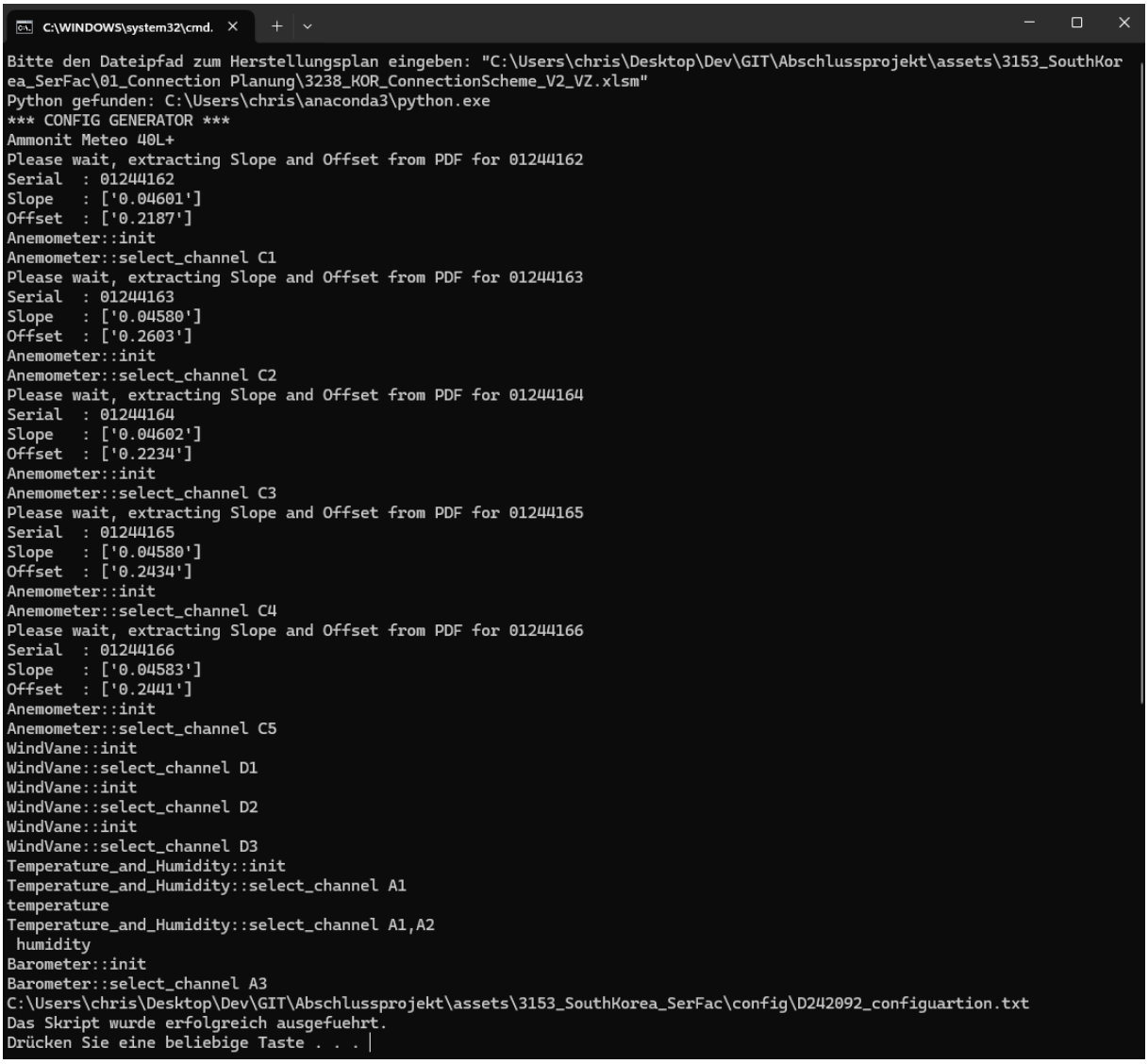

Abbildung 6: Erfolgreiche Ausführung des Programms

```python
def create_config_from_manufacturing_plan(self, manufacture_plan_path) :

    self.validate_path(manufacture_plan_path)
    output_folder_path, certificates_folder = self.create_paths(manufacture_plan_path)
    self.validate_path(certificates_folder)

    sensors_data_frame, project_informations = excel_reader_manufactureplan.read_man_plan(manufacture_plan_path)

    self.projekt_number = project_informations[0]
    self.name = project_informations[1]
    self.serial_number = project_informations[2]
    ##############
    try:
        self.initialize_channels()
    except ValueError as e:
        print("Falscher Datenlogger")
        sys.exit(1)
    self.create_system_info()
    self.create_channels()
    self.create_system_dict()
    ##############

    typ_no = ""
    label = ""
    serial_number = ""
    height = ""
    cert_no = ""
    cert = False
    slope = None
    offset = None
    columns = sensors_data_frame.columns.tolist()
    for index, row in sensors_data_frame.iterrows():
        for x, col in enumerate(columns):
            if col == "serial_number":
                serial_number = row.iloc[x]
            elif col == "type_no":
                typ_no = row.iloc[x]
            elif col == "label":
                label = row.iloc[x]
            elif col == "height":
                height = row.iloc[x]
            elif col == "cert_no":
                if row.iloc[x] != "-":
                    cert_no = row.iloc[x]
                    print(f"Please wait, extracting Slope and Offset from PDF for {serial_number}")
                    slope, offset = pdf_extractor_slope_offset.extract_slope_offset(serial_number, certificates_folder, self.search_values)
                    cert = True
                    if not slope and not offset:
                        print(f"Can't find Slope and Offset for {serial_number} Please add it manually")
        if cert :
            self.add_sensor(typ_no, label, serial_number, height, cert_no, slope, offset)
            cert = False
            slope = None
            offset = None
        else:
            self.add_sensor(typ_no, label, serial_number, height)
    config = self.create_ini_config(output_folder_path)
    return config
```

Abbildung 7: create_config_from_manufacturingplan()

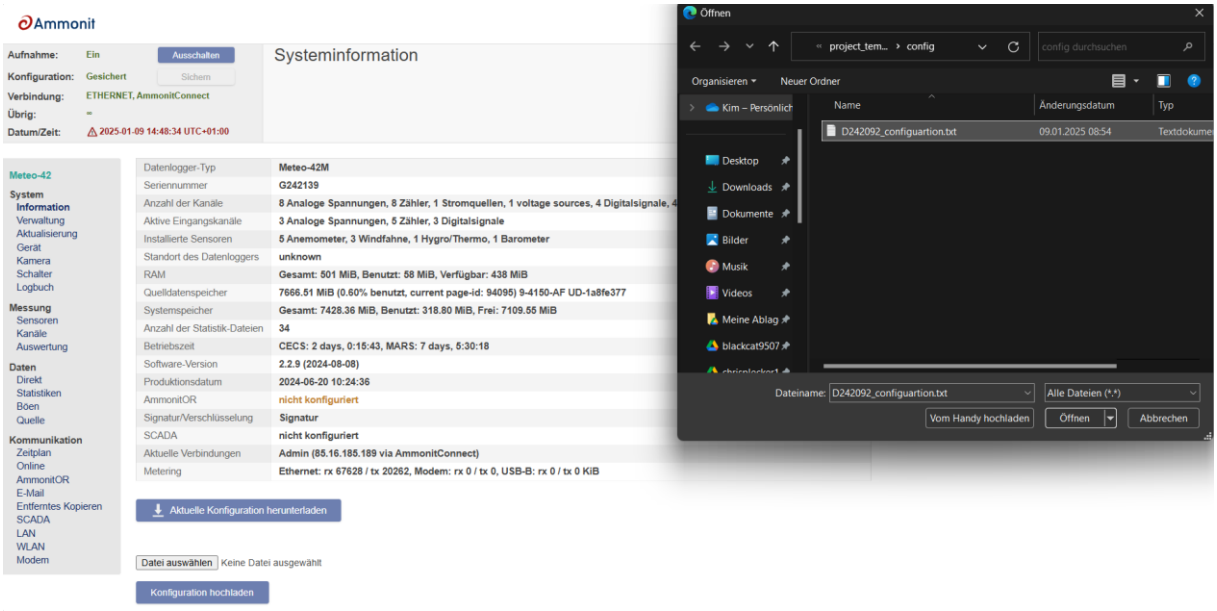

Abbildung 8: Konfiguration auf Ammoniten Aufspielen
(Zu Testzecken auf neuer Datenlogger Generation - Ammonit Meteo 42 – Test Erfolgreich)

A5 Entwickler Doku

`create_paths`(*`manufacture_plan_path`*)

Erstellt die benötigten Ordnerpfade für das Projekt.

Diese Methode nimmt den Pfad zum Herstellungsplan, extrahiert den Projektordner und erstellt erforderliche Unterordner, falls diese noch nicht existieren. Der Hauptordner „config“ wird dabei erstellt, und auch der Ordner für Zertifikate wird ermittelt.

Args:

manufacture_plan_path (str): Der Pfad zum Herstellungsplan.

Returns:

tuple: Ein Tupel bestehend aus zwei Pfaden:

- Der Pfad zum „config“-Ordner im Projektverzeichnis.
- Der Pfad zum „06_Certificates“-Ordner im Projektverzeichnis.

Abbildung 9: 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

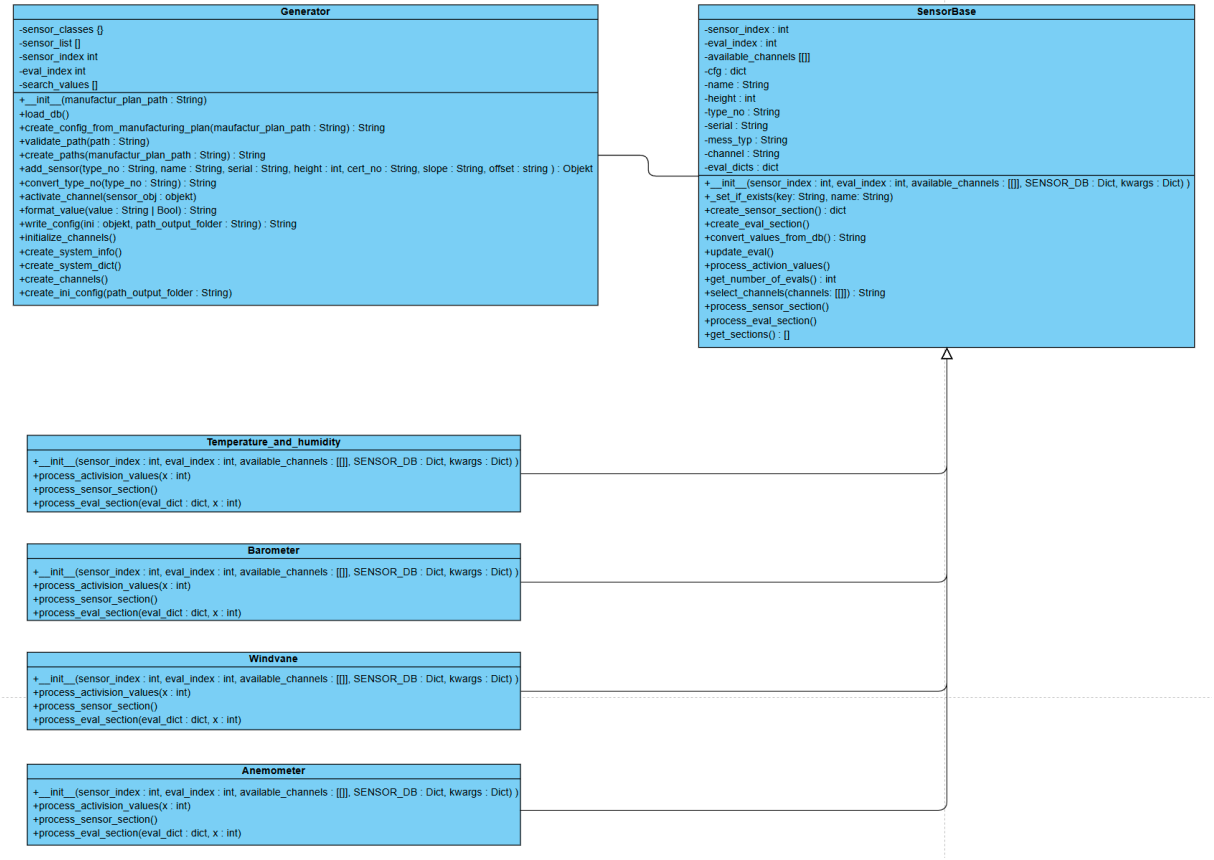

Abbildung 10: Klassendiagramm