Titel: Individuelle Projekt Arbeit

Thema: Testautomation mit Python

Beschreibung: Erstellung eines Import-Scripts, welches auf Excel basierende TsNet-Testscripts in eine JSON-Struktur umwandelt und als JSON-File abspeichert

Key Words: IPA, Import, Python, TsNet, JSON

Speicherort:

Dokument Kategorie:

Revision: 1.0

Änderungsdatum: 02.04.2020

Dokument Status: Beendet

Autor: Timo Gloor

Abteilung:

Verantwortliche Stelle: timo.gloor@siemens.com

Firma:

Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Rev | Datum | Autor | Änderungen |
| 0.1 | 18.02.2020 | Timo Gloor | Status = **In Bearbeitung**   * Strukturaufbau und Gliederung * Tätigkeiten und Meilensteine * Variantenanalyse * Projektorganisation * Einführung * Arbeitsjournal Tag 1 |
| 0.2 | 19.02.2020 | Timo Gloor | Status = **In Bearbeitung**   * Variantenanalyse * Konzept * Excel-Daten Zugriff * Projektorganisation * Arbeitsjournal Tag 2 |
| 0.3 | 20.02.2020 | Timo Gloor | Status = **In Bearbeitung**   * Projektorganisation * Konzept * Kapitel Kontrollieren * Projektorganisation * Arbeitsjournal Tag 3 |
| 0.4 | 24.03.2020 | Timo Gloor | Status = **In Bearbeitung**   * Modultests * Module * Arbeitsjournal Tag 4 |
| 0.5 | 25.03.2020 | Timo Gloor | Status = **In Bearbeitung**   * Arbeitsjournal Tag 5 * Modultest |
| 0.6 | 26.03.2020 | Timo Gloor | Status = **In Bearbeitung**   * Arbeitsjournal Tag 6 * Kapitel Realisieren |
| 0.7 | 27.03.2020 | Timo Gloor | Status = **In Bearbeitung**   * Arbeitsjournal Tag 7 * Kapitel Realisieren * Glossar * Produkttest |
| 0.8 | 31.03.2020 | Timo Gloor | Status = **In Bearbeitung**   * Arbeitsjournal Tag 8 * IPA-Kurzfassung * Kapitel Realisieren * Schlusswort |
| 0.9 | 01.04.2020 | Timo Gloor | Status = **In Bearbeitung**   * Arbeitsjournal Tag 9 * Glossar * Kapitel Realisieren * Tabellenverzeichnis * Abbildungsverzeichnis * Quellenverzeichnis |
| 1.0 | 02.04.2020 | Timo Gloor | Status = **Beendet**   * Arbeitsjournal Tag 10 * Soll-Ist-Vergleich * Formatierung |

Inhaltsverzeichnis

[1. Teil 1 6](#_Toc36730108)

[1.1 Einführung 6](#_Toc36730109)

[1.1.1 IPA 6](#_Toc36730110)

[1.1.2 Zweck des Dokumentes 6](#_Toc36730111)

[1.1.3 Zielpublikum 6](#_Toc36730112)

[1.2 Projektauftrag gemäss PkOrg 7](#_Toc36730113)

[1.2.1 Titel der Arbeit 7](#_Toc36730114)

[1.2.2 Ausgangslage 7](#_Toc36730115)

[1.2.3 Detaillierte Aufgabenstellung 7](#_Toc36730116)

[1.2.3.1 Einführung 7](#_Toc36730117)

[1.2.3.2 Aktuelle Situation, Hintergrund 7](#_Toc36730118)

[1.2.3.3 Entwicklungsumgebung, Rahmenbedingungen 8](#_Toc36730119)

[1.2.3.4 Zusammenfassung Anforderung 9](#_Toc36730120)

[1.2.3.5 Spezifikation 10](#_Toc36730121)

[1.2.3.6 Funktionsbeschreibung 10](#_Toc36730122)

[1.2.3.7 Definition der Ausgangsdaten TestCase Files JSON 11](#_Toc36730123)

[1.2.3.8 Definition der Eingangsdaten TsNet-File XLS 14](#_Toc36730124)

[1.2.3.9 Pseudo-Code der Gesamtfunktion 16](#_Toc36730125)

[1.2.3.10 Daten-Mapping Excel auf JSON 17](#_Toc36730126)

[1.2.3.11 Fehlerbehandlung, Statusmeldungen 20](#_Toc36730127)

[1.2.3.12 Testing und Dokumentation 21](#_Toc36730128)

[1.2.3.13 Organisatorisches 21](#_Toc36730129)

[1.2.4 Mittel und Methoden 22](#_Toc36730130)

[1.2.5 Vorkenntnisse 22](#_Toc36730131)

[1.2.6 Vorarbeiten 22](#_Toc36730132)

[1.2.7 Neue Lerninhalte 22](#_Toc36730133)

[1.2.8 Arbeiten in den letzten 6 Monaten 22](#_Toc36730134)

[1.3 Projektorganisation 23](#_Toc36730135)

[1.3.1 Spezielle Bedingungen 23](#_Toc36730136)

[1.3.2 Arbeitsplatz / Umgebung 23](#_Toc36730137)

[1.3.3 Datensicherung 24](#_Toc36730138)

[1.3.3.1 Dokumente 24](#_Toc36730139)

[1.3.3.2 Sourcecode 24](#_Toc36730140)

[1.3.4 Abteilung SI BP R&D ZG CS SAP 24](#_Toc36730141)

[1.3.5 Beteiligte Personen und Dienste 25](#_Toc36730142)

[1.3.5.1 Personen 25](#_Toc36730143)

[1.3.5.2 Dienste 26](#_Toc36730144)

[1.3.6 Verwendete Projektmethode 27](#_Toc36730145)

[1.3.7 Risikobeschreibung 28](#_Toc36730146)

[1.4 Planung 29](#_Toc36730147)

[1.4.1 Aufbau Zeitplan 29](#_Toc36730148)

[1.4.2 Legende zum Zeitplan 29](#_Toc36730149)

[1.4.3 Soll-Zeitplan 30](#_Toc36730150)

[1.4.4 Ist-Zeitplan 31](#_Toc36730151)

[1.4.5 Soll-Ist-Vergleich 32](#_Toc36730152)

[1.4.6 Tätigkeiten 33](#_Toc36730153)

[1.4.7 Meilensteine 35](#_Toc36730154)

[1.5 Arbeitsjournal 36](#_Toc36730155)

[1.5.1 Zweck des Arbeitsjournales 36](#_Toc36730156)

[1.5.2 Aufbau 36](#_Toc36730157)

[1.5.3 Arbeitsjournale vom 18.03.2020 bis 02.04.2020 37](#_Toc36730158)

[2. Teil 2 47](#_Toc36730159)

[2.1 IPA Kurzfassung 47](#_Toc36730160)

[2.1.1 Ausgangssituation 47](#_Toc36730161)

[2.1.2 Umsetzung 47](#_Toc36730162)

[2.1.3 Ergebnis 47](#_Toc36730163)

[2.2 Entscheiden 48](#_Toc36730164)

[2.2.1 Variantenanalyse 48](#_Toc36730165)

[2.2.1.1 Beschreibung 48](#_Toc36730166)

[2.2.1.2 Vorgehen 48](#_Toc36730167)

[2.2.1.3 Resultat der Recherche 48](#_Toc36730168)

[2.2.1.4 Entscheid 48](#_Toc36730169)

[2.3 Realisieren 49](#_Toc36730170)

[2.3.1 TsNet Import Script 49](#_Toc36730171)

[2.3.1.1 Zweck 49](#_Toc36730172)

[2.3.1.2 Anwendung 49](#_Toc36730173)

[2.3.2 Konzept 50](#_Toc36730174)

[2.3.2.1 Aktivitätsdiagramm 50](#_Toc36730175)

[2.3.2.2 Genereller Aufbau der Applikation 50](#_Toc36730176)

[2.3.2.3 Unterteilung der Module 51](#_Toc36730177)

[2.3.2.4 Fehlerbehandlung und Statusmeldungen 52](#_Toc36730178)

[2.3.2.5 Helfer-Funktionen 53](#_Toc36730179)

[2.3.2.6 Umwandlung der Excel-Daten in eine JSON-Struktur 54](#_Toc36730180)

[2.3.3 Module 55](#_Toc36730181)

[2.3.3.1 Meta-Daten 55](#_Toc36730182)

[2.3.3.2 Connections-Daten 57](#_Toc36730183)

[2.3.3.3 Test-Daten 58](#_Toc36730184)

[2.3.3.4 Einsatz der Module im Code 60](#_Toc36730185)

[2.3.4 Python 61](#_Toc36730186)

[2.3.5 Excel-Daten Zugriff mittels Python-Library 62](#_Toc36730187)

[2.3.5.1 Vorbereitung 62](#_Toc36730188)

[2.3.5.2 Anwendung 62](#_Toc36730189)

[2.3.6 Konventionen im Code 64](#_Toc36730190)

[2.3.6.1 Globale Variablen 64](#_Toc36730191)

[2.3.6.2 Konstanten 64](#_Toc36730192)

[2.3.6.3 Funktionen 64](#_Toc36730193)

[2.3.6.4 Variablen 64](#_Toc36730194)

[2.4 Kontrollieren 65](#_Toc36730195)

[2.4.1 Testkonzept 65](#_Toc36730196)

[2.4.1.1 Testumgebung 65](#_Toc36730197)

[2.4.1.2 Testdaten 66](#_Toc36730198)

[2.4.1.3 Testablauf 66](#_Toc36730199)

[2.4.2 Modultests 67](#_Toc36730200)

[2.4.2.1 «meta»-Daten-Tests 67](#_Toc36730201)

[2.4.2.2 «connection»-Daten-Tests 69](#_Toc36730202)

[2.4.2.3 «test»-Daten-Tests 70](#_Toc36730203)

[2.4.3 Produkttest 72](#_Toc36730204)

[2.5 Auswerten 75](#_Toc36730205)

[2.5.1 Schlusswort 75](#_Toc36730206)

[2.6 Glossar 76](#_Toc36730207)

[2.7 Quellen 78](#_Toc36730208)

[2.8 Anhang 78](#_Toc36730209)

**Tabellenverzeichnis**

[Tabelle 1: Sheet Overview, Range TestCases 17](#_Toc36730210)

[Tabelle 2: Meta Section of JSON File 17](#_Toc36730211)

[Tabelle 3: Connections Section of JSON File 17](#_Toc36730212)

[Tabelle 4: Sheet TestCasexx\_yyyy, Range Header 18](#_Toc36730213)

[Tabelle 5: Sheet TestCasexx\_yyyy, Range Actions 18](#_Toc36730214)

[Tabelle 6: Sheet TestCasexx\_yyyy, Range ObjIn 18](#_Toc36730215)

[Tabelle 7: Eingangsdaten Testcasexx\_yyyy 19](#_Toc36730216)

[Tabelle 8: Sheet TestCasexx\_yyyy, Range Actions 19](#_Toc36730217)

[Tabelle 9: Sheet TestCasexx\_yyyy, Range DataOut 19](#_Toc36730218)

[Tabelle 10: Beteiligte Personen 25](#_Toc36730219)

[Tabelle 11: Verwendete Dienste 26](#_Toc36730220)

[Tabelle 12: Risikobeschreibung 28](#_Toc36730221)

[Tabelle 13: Phasen des Zeitplanes 29](#_Toc36730222)

[Tabelle 14: Legende zum Zeitplan 29](#_Toc36730223)

[Tabelle 15: Tätigkeiten 34](#_Toc36730224)

[Tabelle 16: Meilensteine 35](#_Toc36730225)

[Tabelle 17: Zugriffarten auf eine Zelle 63](#_Toc36730226)

[Tabelle 18: Meta-Daten-Testfälle 68](#_Toc36730227)

[Tabelle 19: Meta-Daten-Testergebnisse 68](#_Toc36730228)

[Tabelle 20: Connections-Daten-Testfälle 69](#_Toc36730229)

[Tabelle 21: Connections-Daten-Testergebnisse 70](#_Toc36730230)

[Tabelle 22: Test-Daten-Testfälle 71](#_Toc36730231)

[Tabelle 23: Test-Daten-Testergebnisse 71](#_Toc36730232)

[Tabelle 24: Test-Daten-Nachtests 72](#_Toc36730233)

[Tabelle 25: Produkttestfälle 73](#_Toc36730234)

[Tabelle 26: Produkttest Ergebnisse 74](#_Toc36730235)

[Tabelle 27: Glossar 77](#_Toc36730236)

**Abbildungsverzeichnis**

[Abbildung 1: Overview TsOpen 8](#_Toc36730237)

[Abbildung 2: Übersicht TsNet-Import 10](file:///C:\Users\z003nn1r\Desktop\!IPA\IPA%20Timo%20Gloor\Dokumentation\IPA-Bericht-TimoGloor.docx#_Toc36730238)

[Abbildung 3: Sheet Config, Range Meta 14](#_Toc36730239)

[Abbildung 4: Sheet Config, Range Devicelist 14](#_Toc36730240)

[Abbildung 5: Sheet Overview, Range TestCases 15](#_Toc36730241)

[Abbildung 6: Sheet TestCasexx\_yyyy 15](#_Toc36730242)

[Abbildung 7: Arbeitsplatz 23](file:///C:\Users\z003nn1r\Desktop\!IPA\IPA%20Timo%20Gloor\Dokumentation\IPA-Bericht-TimoGloor.docx#_Toc36730243)

[Abbildung 8: Ordnerstruktur 24](#_Toc36730244)

[Abbildung 9: Organigramm 25](#_Toc36730245)

[Abbildung 10: IPERKA 27](file:///C:\Users\z003nn1r\Desktop\!IPA\IPA%20Timo%20Gloor\Dokumentation\IPA-Bericht-TimoGloor.docx#_Toc36730246)

[Abbildung 11: Soll-Zeitplan 30](#_Toc36730247)

[Abbildung 12: Ist-Zeitplan (Grüne Felder = Ist-Werte) 31](#_Toc36730248)

[Abbildung 13: Aktivitätsdiagramm 50](#_Toc36730249)

[Abbildung 14: Beispiel Summary-Ausgabe 52](#_Toc36730250)

[Abbildung 15: Beispiel Dictionary 54](#_Toc36730251)

[Abbildung 16: Struktogramm Meta-Daten-Modul 55](#_Toc36730252)

[Abbildung 17: Struktogramm Connections-Daten-Modul 57](#_Toc36730253)

[Abbildung 18: Struktogramm Test-Daten-Modul 58](#_Toc36730254)

[Abbildung 19: Codeaufbau 60](#_Toc36730255)

[Abbildung 20: Python Installation 61](#_Toc36730256)

[Abbildung 21: Python Installation überprüfen 61](#_Toc36730257)

[Abbildung 22: Import xlrd-Library 62](#_Toc36730258)

[Abbildung 23: Erzeugung eines Workbook-Objektes 62](#_Toc36730259)

[Abbildung 24: Erzeugung eines Worksheet-Objektes 63](#_Toc36730260)

[Abbildung 25: Zugriff auf Excel-Zellen 63](#_Toc36730261)

[Abbildung 26: Systeminformationen 65](#_Toc36730262)

Teil 1

## Einführung

### IPA

Die IPA ist ein Projekt, welches jeder Lernende im Bereich der Informatik am Schluss der Lehre absolviert. Ausgeschrieben heisst IPA **I**ndividuelle **P**raktische **A**rbeit. Dieses Projekt dauert 80 Stunden und soll aufzeigen, ob der Kandidat das erforderliche Wissen und Können beherrscht. Das Organisatorische rund um die IPA wird auf der Plattform PkOrg verwaltet.

### Zweck des Dokumentes

Der IPA-Bericht besteht aus zwei Teilen. Im ersten Teil geht es um die generelle Projektdokumentation mit dem Arbeitsjournal, Terminplanung und so weiter. Im zweiten Teil wird das eigentliche Projekt dokumentiert. Mit beiden Teilen zusammen enthält der Bericht die Dokumentation über alle Tätigkeiten, welche vom Kandidaten während des Projektes gemacht wurden.

### Zielpublikum

In erster Linie richtet sich das Dokument vor allem an die Experten und Fachvorgesetzten. Allerdings soll die Programmdokumentation ebenfalls anderen Entwicklern helfen das Programm zu verstehen, so dass später die Applikation allenfalls weiterentwickelt werden kann.

## Projektauftrag gemäss PkOrg

### Titel der Arbeit

Testautomation mit Python

### Ausgangslage

Controller in der Gebäudeautomation beinhalten auch die Applikation, die die eigentlichen Steuer- und Regelaufgaben übernimmt. Diese Applikationen werden in dieser Abteilung entwickelt und getestet. Für den Test der Applikationen wird ein komplett neues Test-Framework entwickelt. Dieses Test-Framework ermöglicht eine weitgehende Automation der Applikationstests. Das Test-Framework beinhaltet auch den „TsNet-Import“. Diese Komponente ermöglicht es, bestehende, mit „TsNet“ gemachte Test-Spezifikationen zu importieren und weiter zu verwenden. Aufgabe der IPA ist es, die „TsNet-Import“-Funktion zu entwickeln.

### Detaillierte Aufgabenstellung

#### Einführung

Controller in der Gebäudeautomation beinhalten auch die Applikation, die die eigentlichen Steuer- und Regelaufgaben übernimmt. Diese Applikationen werden in dieser Abteilung entwickelt und getestet. Für den Test der Applikationen wird ein komplett neues Test-Framework entwickelt. Dieses TestFramework ermöglicht eine weitgehende Automation der Applikationstests. Das Test-Framework beinhaltet auch den „TsNet-Import“. Diese Komponente ermöglicht es, bestehende, mit „TsNet“ gemachte Test-Spezifikationen zu importieren und weiter zu verwenden. Aufgabe der IPA ist es, die „TsNet-Import“-Funktion zu entwickeln.

#### Aktuelle Situation, Hintergrund

Für den Test von Applikationen wird ein komplett neues Test-Framework „TsOpen“ entwickelt. Dieses Test-Framework ermöglicht eine weitgehende Automation der Applikationstests. Das Test-Framework beinhaltet auch den „TsNet-Import“. Diese Komponente ermöglicht es, bestehende, mit „TsNet“ gemachte Test-Spezifikationen zu importieren und weiter zu verwenden. Aufgabe der IPA ist es, diese „TsNet-Import“-Funktion zu entwickeln.

Übersicht TsOpen Test-Framework

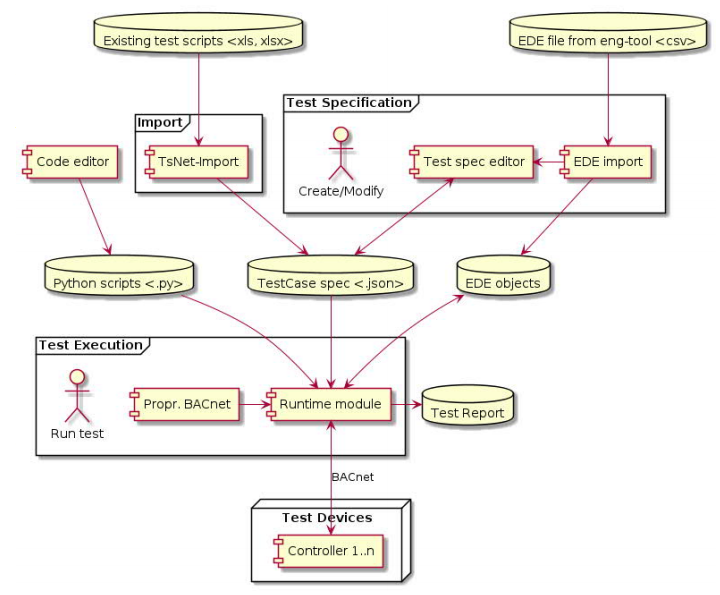


Abbildung : Overview TsOpen

Obwohl prinzipiell verschiedene Test Execution Engines geplant sind, wird zunächst nur eine Test Execution Engine entwickelt, die mit Hilfe von PYTHON über BACnet-Kommunikation mit dem Controller Daten von Objekten lesen und schreiben kann.

#### Entwicklungsumgebung, Rahmenbedingungen

Entwicklungsumgebung vorgeschrieben:

* Python Release 3.7 oder höher
* MICROSOFT Visual Studio Code
* GIT im Rahmen von code.siemens.com
* Notebook mit MICROSOFT Windows 10
* Frei wählbares Graphikprogramm zur Programmdokumentation, Empfehlung : PlantUML  
  <https://plantuml.com/de/>
* MICROSOFT Office zum Erstellen der Produkt- und Projektdokumentation
* Empfehlung: Notepad++ Programmeditor mit JSON tool, <https://notepad-plus-plus.org/>

Weiterhin werden folgende Tools empfiohlen

* Graphikprogramm zur Programmdokumentation, PlantUML  
  <https://plantuml.com/de/>
* Notepad++ Programmeditor mit JSON tool, <https://notepad-plus-plus.org/>

Aufgrund der Eingangsdaten muss der TsNet-Import die Excel-Daten (.xls) lesen können. Idealerweise kann das direkt vom Python-Programm ohne Rückgriff auf Excel-Funktionen erfolgen. Entsprechende Libraries sollen gesucht und ausprobiert werden.

Sollte das nicht möglich sein, kann auch ein Exportieren von Excel in ein Standard-Format (csv, html, xml…) eingesetzt werden. Wenn diese Variante gewählt wird, ist das zu begründen.

Vor Beginn der IPA werden noch mehrere Eingangsdateien geliefert, die zum Testen in der IPA verwendet werden.

#### Zusammenfassung Anforderung

Der Kandidat soll im Rahmen der IPA Programmcode und einen IPA-Bericht erstellen. Dabei soll er folgende Anforderungen erfüllen:

* Variantenanalyse: Auswahl und Kurztest einer Python-Library zum Einlesen von Excel-Dateien (.xls). Falls nicht möglich auf csv, html oder xml ausweichen, Entscheidungen müssen im Rahmen des IPA-Bericht dokumentiert sein. Siehe dazu Rahmenbedingungen, Kapitel 2.
* Gesamtfunktion: Die zu realisierende Gesamtfunktion ist als Pseudocode dargestellt. Der Kandidat erstellt zunächst ein Konzept zur Umsetzung und dieses wird im Rahmen des IPA-Berichts dokumentiert, evtl. in passenden UML-Diagrammen (Sequenz-, Interaktionsdiagramm). Siehe dazu Pseudo-Code der Gesamtfunktion, Kapitel 4.4
* Benutzerinterface: TsNet import wird per Kommandozeile vom User gestartet und hat keine Menüinteraktionen sonst. Siehe Kapitel 4.1 Funktionsbeschreibung.
* Mapping: Datenmapping von Excel zu JSON gemäss Kapitel 4.5 umgesetzt. Das Zielformat JSON ist in Kapitel 4.2 und das Quellformat Excel ist in Kapitel 4.3 beschrieben.
* Fehlerbehandlungen und Statusmeldungen: sind gemäss Kapitel 4.6 abgehandelt.
* Testing: ist im Kapitel 5 beschrieben. Die Testspezifikation und der Testbericht sind im IPA-Bericht zu dokumentieren.
* Programmdokumentation: Ist im Kapitel 5 beschrieben und im IPA-Bericht zu dokumentieren.
* Projektdoumentation: Da der Kandidat eigenverantwortlich dieses Projekt abarbeitet, ist auch eine Projektdokumentation im Rahmen des IPA-Berichts zu erstellen.
* Fremde Hilfe: Ist in der Projektdokumentation darzustellen.
* Fremder Code: Es muss klar ersichtlich sein ob Programmteile selbst erstellt wurden oder ob sie schon vorhanden waren oder aus einem Framework stammen.

#### Spezifikation

Die Spezifikation besteht aus:

* Funktionsbeschreibung
* Beschreibung der Ausgangsdaten
* Beschreibung der Eingangsdaten
* Beschreibung der Gesamtfunktion als Pseudo-Code
* Mapping-Tabellen zwischen Ausgangsdaten und Eingangsdaten.

2

Sprechblasen mit Nummer verbinden die einzelnen Teile der Spezifikation miteinander.

So bezeichnet (2) die Projekt-Meta-Daten in den Eingangsdaten, in den Ausgangsdaten, im Pseudo-Code und in den Mapping-Tabellen.

<var> kennzeichnet Einträge mit dem Wert der Variable var

„const“ kennzeichnet Einträge mit einer Konstanten

#### Funktionsbeschreibung

Task of TsNet import is to re-use existing test specifications for the new test environment TsOpen.

TsNet import shall read the existing test specifications  
from TsNet V1 as \*.xls files.

(Import of TsNet V2 \*.xlsx files is not part) of the IPA.

TsNet import will be called by the user as command line program.  
**-> TsNetImport SourceFile DestinationFolder**

TsNet import does not have a menu for user interaction.

It only prints any errors caused by file handling to the command line. TsNet can be handled by external scripts to allow sequential conversion of multiple test specifications.

The Excel file is read by TsNet Import and interpreted.  
For each test case sheet, a JSON file is created. These JSON files use a syntax specification which is independent from the test environment and the test tool used.  
Apart from meta information and comments, JSON uses function calls to influence data on the test device and to compare data in the test device with expected values.  
During test execution, these function calls are executed by the RunTime component, using the possibilities of the actual test environment to interact with the test device.

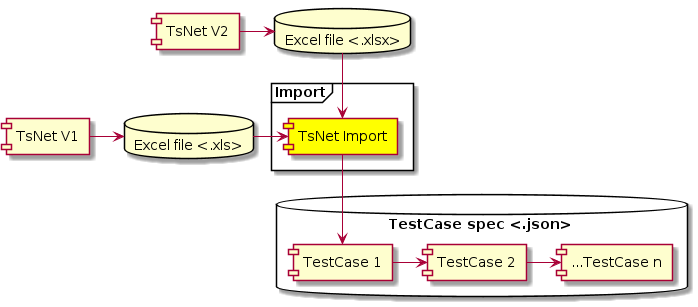


Abbildung : Übersicht TsNet-Import

#### Definition der Ausgangsdaten TestCase Files JSON

Innerhalb einer Test-Spezifikation wird für jeden Testfall ein JSON File verwendet.

Das JSON File hat verschiedene Bereiche, die neben dem eigentlichen Testfall auch allgemeine Daten beinhalten, die zum Abarbeiten des Tests erforderlich sind.

Das aktuell gültige JSON-File findet sich im Intranet unter: <https://code.siemens.com/testingtoolbox/TsOpen-testcase-json/blob/master/testcase-OSS-testplatform.json>

Diese Version ist für die IPA zu verwenden.

Der Bereich „meta“ beinhaltet allgemeine Daten zum Testfall

Der Bereich „connections“ beinhaltet Verbindungsdaten zu den Controllern, die für den Test verwendet werden.

Der Bereich „test“ beinhaltet die Beschreibung des Testfalls mit den Testschritten.

Der Bereich „simulation“ ermöglicht die Definition von Daten zur Prozesssimulation. Dieser Bereich bleibt leer, da die bestehenden TsNet-Tests keine Prozesssimulation ermöglicht haben.

Der Bereich „log“ ermöglicht das Loggen von Prozessdaten während des Tests. Dieser Bereich bleibt leer, da die bestehenden TsNet-Tests kein Logging ermöglicht haben.

**Struktur des JSON-Files**

{

"meta": {"testData", "testObject" },

"connections": [ {connection}, {connection}…} ],

"test": [ {"stepName“, "stepList": [ {action}, {action},…] }, {"stepName“, "stepList": [ {action}, {action},…] }…]

"simulation",

"log"

}

Beispiel für ein JSON File mit Daten passend zum Excel-Sheet in dieser Spezifikation

{

2

"meta": {

"testData": {

"projectName": "TsNetV1\_Example",

"testName": "TopAF Lgt03",

"testDescription": "Test imported from TsNet“

"testCaseName": "Open Space Office",

"testCaseDescription": " Set all relevant BACnet objects to initial values",

"version": "1.02",

"author": "KryenbuA",

"date": "23.07.2015",

"type": "PT",

"automation": "automatic"

},

"testObject": {

"name": "Lgt03",

"type": "TopAF",

"description": " Test Scenarios (see https://workspace.sbt.siemens.com/content/10003162/team\_appl/Documents/80\_Developing/02\_Charts/04\_Lighting/00%20Misc/Light\_Scenarios\_0v6\_RD.doc)",

"version": "2.412",

"source": "Share",

"path" : ""

}

},

3

"connections": [

{

"deviceNo": 1,

"deviceName": "AS\_1",

"deviceAddress": "192.168.251.1",

"deviceType": " DXR2.E12P-1"

},

{

"deviceNo": 2,

"deviceName": "AS\_2",

"deviceAddress": "192.168.251.2",

"deviceType": " DXR2.E12P-1"

}

],

"test": [

4

{

"stepName": " Switch off/unoccupied/dark",

5

"stepList": [

6

7

{

"func": "Set", "obj": "AS\_1.LgtBtn(1)", "prop": "PrVal", "val": (6,0,0)

},

{

"func": "Set", "obj": "AS\_1.Brgt", "prop": "PrVal", "val": 0

},

{

"func": "Set", "obj": "AS1.PscDet(1)", "prop": "PrVal", "val": 0

},

{

10

"func": "Wait", "val": 25

},

9

8

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrgsVal", "val": [0,2],

},

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrPrio", "val": 13,

},

{

"func": "Verify", "obj": "AS\_1.LgtEei", "prop": "PrVal", "val": 5,

}

],

{

"stepName": " Occupied",

"stepList": [

{

"func": "Set", "obj": "AS1.PscDet(1)", "prop": "PrVal", "val": 1

},

{

"func": "Wait", "val": 3

},

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrgsVal", "val": 100,

},

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrPrio", "val": 15,

},

{

"func": "Verify", "obj": "AS\_1.LgtEei", "prop": "PrVal", "val": 5,

}

]

}

// etc until last line …

{

"stepName": " Switch off",

"stepList": [

{

"func": "Set", "obj": "AS1.LgtBtn(1)", "prop": "PrVal", "val": (6,0,0)

},

{

"func": "Wait", "val": 2

},

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrgsVal", "val": 0,

},

{

"func": "Verify", "obj": "AS\_1.LgtCmd(1)", "prop": "PrPrio", "val": 13,

} ]

},

{

"stepName": " End Test"

}

]

}

#### Definition der Eingangsdaten TsNet-File XLS

Das Excel-File besteht aus mehreren Sheets:

Sheet „Config“ beinhaltet allgemeine Daten und Verbindungsdaten zu den Controllern.

Sheet „Overview“ beinhaltet eine Tabelle, in der die einzelnen Testfälle aufgelistet sind.

Ein oder mehrere Sheets „Testcasexx\_yyyy“ beinhalten jeweils einen Testfall mit Nummer xx und Name yyyy, wie in Sheet Overview dargestellt.

Manche Bereiche der Excel-Sheets haben eine dynamische Grösse, die je nach Testfall unterscheidlich ist. Bereiche (Range) sind zur Erklärung mit einem Namen versehen. Falls Bereiche eine dynamische Grösse haben, ist Anfang und Ende eines Bereichs durch klar erkennbare Referenzfelder definiert.

Struktur des Excel-Files:

**Sheet Config**

**Range Meta** beinhaltet Meta-Daten für den gesamten Test. Er geht von A1 bis E16.

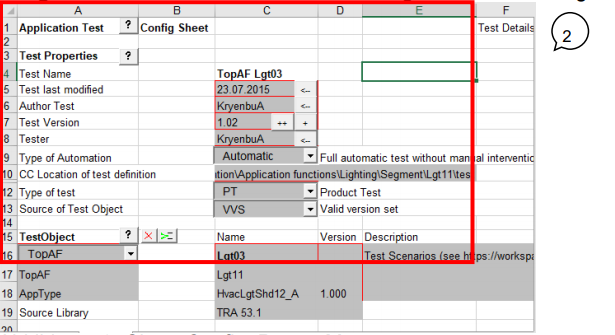


Abbildung 3: Sheet Config, Range Meta

**Range DeviceList** beinhaltet Verbindungsdaten zu den Controllern. Er startet in Spalte A-D, Zeile mit Inhalt „Device-No“ + 1 und geht bis zur nächsten leeren Zelle in Spalte A. Die Anzahl Zeilen ist variabel.

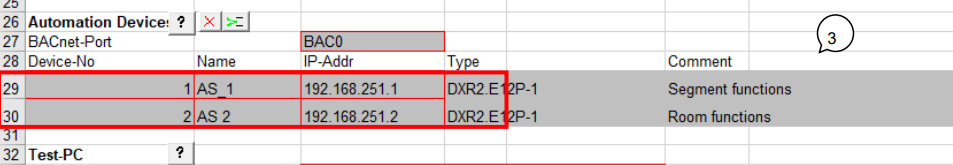


Abbildung 4: Sheet Config, Range Devicelist

**Sheet Overview:**

**Range TestCases** beinhaltet eine Liste aller Testfälle. Er startet in Spalte A-C, Zeile mit dem Inhalt „QC Step number“ + 2 und geht bis zur nächsten leeren Zeile in Spalte A.

Hinweis: Step 1 dient nur als Zusammenfassung und wird ignoriert. Die Liste fängt erst mit Step 2 an.

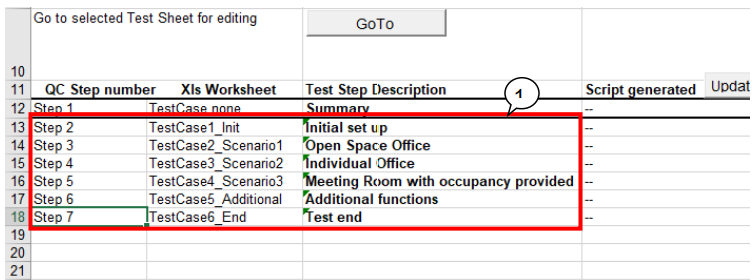


Abbildung 5: Sheet Overview, Range TestCases

**Sheet Testcasexx\_yyyy:** (Name aus Spalte B Sheet „Overview“)  
Das Excel-Sheet teilt sich in folgende Ranges auf:  
**Range Header** beinhaltet testfall-spezifische Informationen: A2..A3  
**Range Actions** beinhaltet die Tests eines Testfalls. Spalte A-C, von der Zeile wo in Spalte A das Feld mit dem Inhalt „Action“ ist + 1, bis zum ersten leeren Feld in Spalte A. Range Actions hat eine flexible Anzahl Zeilen.  
**Range ObjIn** Liste der zu schreibenden Objekte. Von der Spalte, wo in Zeile 2 der Inhalt „In“ steht + 1 bis zur Spalte, wo in Zeile 2 der Inhalt „Out“ steht -1.Zeile 2 bis 16. Range ObjIn hat eine flexible Anzahl Spalten.  
**Range ObjOut** Liste der zu lesenden Objekte. Von der Spalte, wo in Zeile 2 der Inhalt „Out“ steht + 1 bis zu der Spalte, wo in Zeile 2 der Inhalt „End“ steht -1. Zeile 2 bis 16. Range ObjOut hat eine flexible Anzahl Spalten.  
**Range DataIn** Liste der zu schreibenden Daten. Spalte wie ObjIn, Zeile wie in Actions. Range DataIn hat eine Flexible Anzahl an Spalten und Zeilen  
**Range DataOut** Liste der zu erwartenden Werte beim Lesen. Spalte wie in ObjOut, Zeile wie in Actions. Range DataOut hat eine Flexible Anzahl an Spalten und Zeilen.

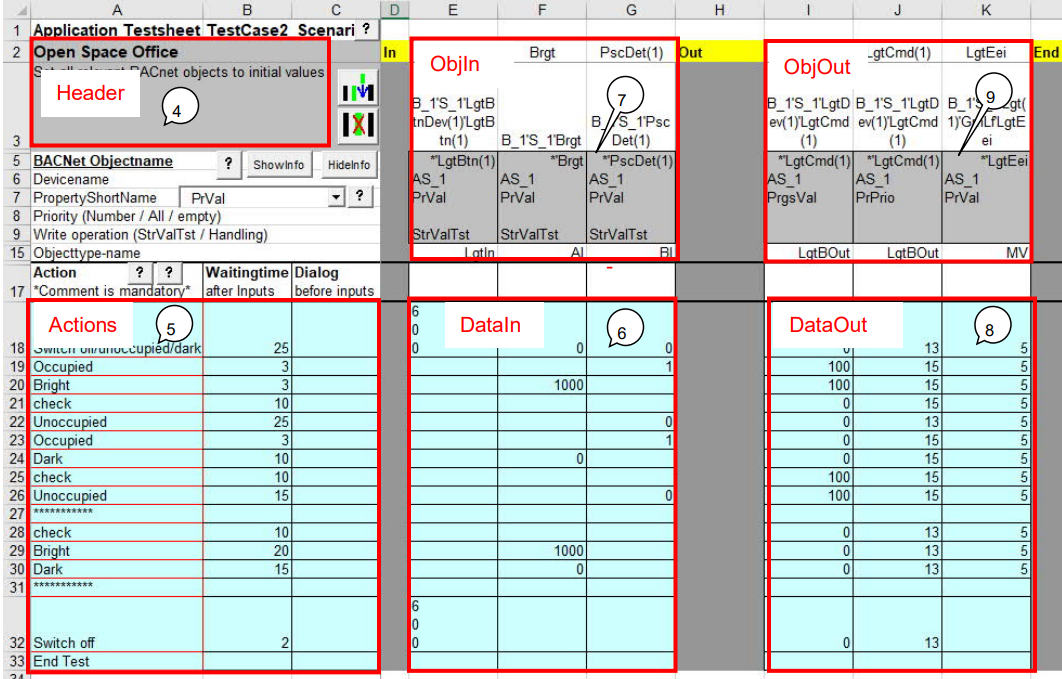


Abbildung : Sheet TestCasexx\_yyyy

#### Pseudo-Code der Gesamtfunktion

**Achtung: Diese Übersicht dient ausschliesslich zur Verdeutlichung der Aufgabenstellung.**

Sie soll keinen Hinweis für die Programmierung darstellen. Sie erhebt keinen Anspruch auf Vollständigkeit, und gibt nicht die Struktur des Programms und der Module wieder. Die Erarbeitung einer sinnvollen Programmstruktur und die Modularisierung des Programms liegt in der Verantwortung des Kandidaten.

//\*\*\* 1. Gesamtfunktion\*\*\*

1

For each line in Sheet „Overview“, Range „TestCases“

Read column B „Sheetname“

Create a new JSON file with name „Sheetname“

2

// 2. Meta Data

Read data from Sheet „Config“ Range „Meta“ and write „meta“ section of JSON file

// 3. Devicedata

For each line in sheet „Config“, Range „Devicelist“

3

Read data and write a „connection“ to the „connections“ section of JSON file

Next line

// 4. Testfall-Daten

4

With Sheet „Sheetname“

Read Range „Header“ and write „testData“ to „meta“ section

// 5. Testschritte

For each line in range „Actions“  
 Create a new entry in „test“ section of JSON file

5

Write „StepName“ in „test“ section of JSON file

Create empty „stepList“ in „test“ section of JSON file

If „Dialog“ is non-empty then write a „verifyByUser“ command to step list

// 6. Aktionen

For each non-empty column in range „DataIn“

Read Devicename, Object, property, prio from range „ObjIn“

7

Read WriteOperation from from range „ObjIn“

If WriteOperation = „Handling“ then write a „suppressFail“ to stepList

Read data of actual line in range „DataIn“

6

Write a „set“ command to step list

Next column

10

If „WaitingTime“ is non-empty then write a „wait“ command to step list

8

// 8. Verifikationen

For each non-empty column in range „DataOut“

9

Read Object, property, prio from range „ObjOut“

Read data of actual line in range „DataIn“ or „DataOut“

Write a „verify“ command to step list

Next column

Next line in ActionList

EndWith sheet

Create empty „simulation“ section in JSON file

Ceate empty „log“ section

Next line in TestCases

#### Daten-Mapping Excel auf JSON

**1. Gesamtfunktion**

For each line in Sheet „Overview“, Range „TestCases“

Read column B „Sheetname“

Create a new JSON file with name „Sheetname“

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten  Excel-File TsNet V1, Sheet “Overview“, Range „Testcases“ | | Ausgangsdaten  1  JSON File | | |
| Spalte | Bedeutung | Bereich | Name | Regel |
| B | Sheetname |  |  | Filename of JSON file |

Tabelle 1: Sheet Overview, Range TestCases

**2. Meta Data**

Read data from Sheet „Config“ Range „Meta“ and write „meta“ section of JSON file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  Sheet “Config“, Range „Meta“ | | Ausgangsdaten JSON  Section „meta“  2 | | |
| Spalte/Zeile | Bedeutung | Bereich | Name | Regel |
|  |  | testData | projectName | Filename of xls-file ohne extension, z.B. TsNetV1\_Example |
|  |  |  | testDescription | „Test imported from TsNet“ |
| C4 | Test Name |  | testName |  |
| C7 | Test Version |  | version |  |
| C6 | Author Test |  | author |  |
| C5 | Date last modified |  | date |  |
| C12 | Type |  | type |  |
| C9 | Type automation |  | automation |  |
| C16 | Test object | testObject | name |  |
| A16 | Objecttype |  | type |  |
| E16 | Description |  | description |  |
| D16 | Version |  | version |  |
| C13 | Source |  | source |  |

Tabelle 2: Meta Section of JSON File

**3. Devicedata**

For each line in sheet „Config“, Range „Devicelist“

Read data and write a „connection“ to the „connections“ section of JSON file

Next line

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  Sheet “Config“, Range „Devicelist“ | | Ausgangsdaten JSON  Section „Connections“  3 | | |
| Spalte | Bedeutung | Bereich | Name | Regel |
| A | Device-No |  | deviceNo |  |
| B | Name |  | deviceName |  |
| C | IP-Addr |  | deviceAddress |  |
| D | Type |  | deviceType |  |

Tabelle 3: Connections Section of JSON File

**4. Testfall Daten**

With Sheet „Sheetname“

Read Range „Header“ and write „testData“ to „meta“ section

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ Range „Header“ | | Ausgangsdaten JSON  section meta  4 | | |
| Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
| A2 | Name | testData | testCaseName |  |
| A3 | Description |  | testCaseDecription |  |

Tabelle 4: Sheet TestCasexx\_yyyy, Range Header

**5. Testschritte**

For each <line> in range „Actions“  
 Create a new entry in „test“ section of JSON file

Write „StepName“ in „test“ section of JSON file

Create empty „stepList“ in „test“ section of JSON file

If „WaitingTime“ is non-empty then write a „wait“ command to step list

If „Dialog“ is non-empty then write a „verifyByUser“ command to step list

5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ Range „Actions“ | | Ausgangsdaten JSON  Section test | | |
| Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
| <line> A | Action |  | stepName |  |
| <line> C | Dialog | stepList | funct = „verifyByUser“  message = <Dialog> | Only when Dialog not empty |

Tabelle 5: Sheet TestCasexx\_yyyy, Range Actions

**6. / 7 Aktionen**

For each <line> in range „Actions“

For each non-empty <column> in range „DataIn“

Read Devicename, Object, property, prio from range „ObjIn“

Read WriteOperation from from range „ObjIn“

If WriteOperation = „Handling“ then write a „suppressFail“ to stepList

Read data of actual line in range „DataIn“

Write a „set“ command to step list

Next <column>

If „WaitingTime“ is non-empty then write a „wait“ command to step list

Next <line>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ Range „ObjIn“ | | Ausgangsdaten JSON  7 | | |
| Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
| 9 <column> | WriteOperation | stepList | funct = „suppressFail“ | Only if WriteOperation = „Handling“ |

Tabelle 6: Sheet TestCasexx\_yyyy, Range ObjIn

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ | | Ausgangsdaten JSON  6 | | |
| Range / Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
|  |  | stepList | Funct = „set“ | Create new function |
| Range “DataIn“, <line>, <column> | Value |  | val = <Value> | Format siehe unten |
| Range „ObjIn“, 5, <column> | Objectname |  | Obj = <Objectname> & <Devicename> | Format siehe unten |
| Range „ObjIn“, 6, <column> | Devicename |  |
| Range „ObjIn“, 7, <column> | Property Shortname |  | Prop = <PropertyShortname> |  |
| Range „ObjIn“, 9, <column> | Priority |  | Prio = <Priority> | only if priority not empty |

Tabelle : Eingangsdaten Testcasexx\_yyyy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ Range „Actions“ | | Ausgangsdaten JSON  10 | | |
| Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
| <line> B | Waitingtime |  | funct = „wait“  time = <Waitingtime> | Only when Waitingtime not empty |

Tabelle 8: Sheet TestCasexx\_yyyy, Range Actions

**8./9. Verifikationen**

For each <line> in range „Actions“

For each non-empty <column> in range „DataOut“

Read Devicename, Object, property, prio from range „ObjOut“

Read data of actual line in range „DataIn“ or „DataOut“

Write a „verify“ command to stepList

Next <column>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingangsdaten Excel  sheet “TestCasexx\_yyy“ | | Ausgangsdaten JSON  9 | | |
| Range / Zeile / Spalte | Bedeutung | Bereich | Name | Regel |
|  |  | stepList | Funct = „verify“ | Create new function |
| Range “DataOut“, <line>, <column> | Value |  | val = <Value> | Format siehe unten |
| Range „ObjOut“, 5, <column> | Objectname |  | Obj = <Objectname> & <Devicename> | Format siehe unten |
| Range „ObjOut“, 6, <column> | Devicename |  |
| Range „ObjOut“, 7, <column> | Property Shortname |  | Prop = <PropertyShortname> |  |
| Range „ObjIn“, 9, <column> | Priority |  | Prio = <Priority> | only if priority not empty |

Tabelle 9: Sheet TestCasexx\_yyyy, Range DataOut

Hinweis zum Format:

val für funct = „set“ kann folgende Formate annehmen:

* Numerisch z.B. 1 , 2.3, 0

Excel verwendet sowohl 2.3 als auch 2,3.   
JSON verwendet 2.3

* Struktur z.B. 6,0,0  
  In Excel wird <ALT><RETURN> bei der Eingabe als Trennzeichen verwendet.   
  JSON verwendet Komma und Klammern ( 6,0,0)

val für funct = „verify“ kann zusätzlich folgende Formate annehmen:

* Bereich numerisch z.B. 1..15 oder 2.3..2.5  
  Excel verwendet 2 Punkte als Trennzeichen  
  JSON verwendet Klammern und Komma [1,15] oder [2.3 , 2.5]
* Bereich Struktur z.B.   
  5..7  
  -1..1  
  0..0  
  In Excel wird <ALT><RETURN> bei der Eingabe als Trennzeichen zwischen den Elementen der Struktur verwendet, und 2 Punkte als Bereich

JSON verwendet Klammern und Komma für Bereich und Struktur ( [5 , 7] , [-1 , 1] , [0 , 0] )

Objectname in Excel kann folgende Formate annehmen

* Voller Name, z.B. B\_1'R\_1'RGrnLf'RLgtEei  
  wird im JSON gleich übernommen
* Name mit Wildcard am Anfang, z.B. \*'LgtEei  
  wird im JSON ohne \* und ' übernommen, z.B. LgtEei
* Name mit Wildcard irgendwo, z.B. \*\_1\*i  
  Führt zu einer Warnmeldung mit Angabe von Excel-Sheet und Zelle, z.B.   
  Warn: Object not identified Sheet „TestCase1Init“ Cell „E5“

Objectname und Devicename werden im JSON zu einem Feld „obj“ kombiniert mit . als Trennzeichen, z.B. AS\_1.LgtEei.

#### Fehlerbehandlung, Statusmeldungen

Bei Ausführung des Programms können diverse Fehler auftreten. Z.B.

- Filehandling

Sourcefile nicht vorhanden, Sourcefile ist kein TsNet V1 File, Speicherplatzproblem, kein Schreibzugriff etc  Fehler

Destination Folder nicht vorhanden  Info, Folder neu anlegen

File mit gleichem Namen im Destination Folder bereits vorhanden  Warnung, überschreiben

- Fehler in Sourcefile

Bereich/Range nicht erkennbar, TestCase Sheet nicht vorhanden  Fehler

Fehlende Daten, die im JSON File gebraucht werden  Warnung, wenn im Bereich „meta“

Fehlende Daten, die im JSON File gebraucht werden  Fehler, wenn im Bereich „connections“ oder „test“

Objectname Wildcard-Problem wie in 2.5.7. beschrieben  Warnung

Bei Fehlern wird das Generieren des JSON-File abgebrochen, und das File gelöscht. Sofern möglich, wird mit dem nächsten Testfall weitergemacht.

Fehler, Warnungen und Infos werden dem Anwender angezeigt.

Vom Betriebssystem erkannte Fehler werden ohnehin in der direkt in der Kommandozeile angezeigt. Alle logischen Fehler werden ebenfalls dem Anwender gemeldet. Die Meldung beinhaltet, sofern möglich

- Art der Meldung (ERR, WARN, INFO)

- Fehlertext/ Infotext (z.B. Sheet „TestCase1\_Init“ not existing)

- Stelle, wo der Fehler auftritt (File Test.xls, Sheet Overview, Zelle B13)

- Folgen (z.B. Continuing..)

Als Info wird dem Anwender jedes fehlerfrei oder mit Warnung angelegte JSON-File angezeigt,

z.B. INFO: Testcase2\_Scenario1 JSON file created

Es wird eine Zusammenfassung angezeigt, z.B. INFO: 4 of 5 JSON files correctly created

#### Testing und Dokumentation

**Modultest**

Während der Implementierung, nach Fertigstellung einer Komponente, ist vom Entwickler ein Modultest durchzuführen. Dieser ist als White-Box-Test zu gestalten. Ziel des Modultests ist es, Fehler sofort festzustellen, und zu korrigieren, so dass die Weiterverwendung des Moduls möglich ist. Vorgaben vom Modultest kommen vom Entwickler selbst. Je nach implementiertem Code können interne und externe Schnittstellen verwendet werden. Nach erfolgreichem Abschluss des Modultests muss dieser dokumentiert werden. Dabei sind die durchgeführten Tests stichprobenartig darzustellen.

**Produkttest**

Nach Ende der Implementation ist ein Produkttest durchzuführen. Dieser ist als Black-Box-Test zu gestalten. Ziel ist es, die Qualität des Produkts sicherzustellen. Dazu sind die definierten externen Schnittstellen zu verwenden, also hier die Eingangsdatei und die Ausgangsdateien sowie die Benutzeroberfläche, d.h. die Fehler-, Warn- und Statusmeldungen. Die Eingangsdateien für diesen Test werden beigestellt. Die Ausgangsdateien werden mit Hilfe eines Programmeditors manuell geprüft, und, falls bereits verfügbar, wird eine erste Version des Runtime-Moduls verwendet werden

**Programmdokumentation**

Die Programmdokumentation soll es ermöglichen, dass andere Fachpersonen

- an der IPA weiterarbeiten können

- später gefundene Fehler korrigieren können

- den Code für ähnliche Aufgabenstellungen wiederverwenden können

- weitere Dokumentation, z.B. Benutzerdokumentationen anfertigen können.

Die Programmdokumentation muss es einer Fachperson ermöglichen, von der Aufgabenstellung bis zu einer Stelle im Code gelangen zu können, um ihn zu modifizieren oder zu testen. Dazu dienen neben Beschreibungen auch zweckmässige graphische Darstellungen in Form von NassiShneidermann-Diagrammen oder UML-Diagrammen aller Art. Die Dokumentation muss auch nachvollziehbar darstellen, warum bei der Entwicklung bestimmte Entscheidungen getroffen worden sind, um z.B. Irrwege nicht mehrmals zu gehen.

#### Organisatorisches

**Projekt Organisation**

Siehe PKOrg [www.pkorg.ch](http://www.pkorg.ch)

**Termine, Kosten**

Termine siehe PKOrg [www.pkorg.ch](http://www.pkorg.ch)

Die Kosten sind durch die Terminvorgabe definiert.

**Erwartete Resultate**

Für alle neuen und geänderten Komponenten:

- Lieferung als Python File im Projektordner

- Lieferung von dokumentiertem und versioniertem Source-Code

- Lieferung der Programmdokumentation (je nach Bedarf Spezifikation, Realisierungsdokumentation, Variantenanalyse, Code-Listing mit Kommentaren o.ä.)

- Dokumentation der Testfälle und Testergebnisse

- Lieferung der Projektdokumentation (Journal, Terminplanung, Projektstatus alle 2 Tage)

### Mittel und Methoden

siehe PDF-File "IPA Timo Aufgabenstellung\_0.4" 10.03.2020 Kapitel 2

### Vorkenntnisse

Der Kandidat hat bereits mehrere Monate Erfahrung in unserer Abteilung mit Python und mit Testautomation in unserem Umfeld. Er hat bereits die komplette Arbeitsumgebung installiert und benutzt. Die vorgegebenen und die empfohlenen Tools sind ihm bekannt Er hat auch bereits ein kleineres Projekt mit Planung, Journal und Programmdokumentation durchgeführt als Vorbeeitung zur IPA.

### Vorarbeiten

Siehe auch Vorkenntnisse. Der Kandidat hat sich bereits mit dem Tool TsNet V1 vertraut gemacht und kennt den Aufbau der entsprechenden Excel-Datei, die seine IPA als Eingangsdatei verwendet.

### Neue Lerninhalte

keine.

### Arbeiten in den letzten 6 Monaten

- Arbeitsumgebung für TsNet (bisheriges Test-Framework) aufgesetzt und Erfahrungen gesammelt - Fehlersuche im TsNet - Mitarbeit an Konzept des "neuen" Test-Frameworks für Applikationstests - Erstellen einer Anleitung zum Einsatz eines Python Log-Moduls - Erstellung einer Anleitung für den Einsatz von Python zur Kommunikation zum Controller

## Projektorganisation

### Spezielle Bedingungen

Normalerweise wird die IPA im Betrieb während zehn Arbeitstagen durchgeführt und die Meetings vor Ort in einem Meetingraum gehalten. Da zur Zeit der Durchführung der IPA (März/April 2020) eine Pandemie durch den COVID-19-Virus herrscht, hat die Siemens Schweiz AG alle Mitarbeiter dazu aufgefordert die Arbeit vom Home-Office aus zu erledigen. Aufgrund dieser Massnahme wird die Durchführung der IPA von zu Hause aus erfolgen. Die Meetings können trotzdem via „Circuit“ gehalten werden. Dieses Tool ermöglicht Video- und Sprachkonferenzen, bei dem ebenfalls der Bildschirm mit den einzelnen Teilnehmern geteilt werden kann.

### Arbeitsplatz / Umgebung

Abbildung : Arbeitsplatz

Wie bereits beschrieben, findet die IPA von zu Hause aus statt. Der Laptop wurde von der Siemens Schweiz zur Verfügung gestellt. Es handelt sich dabei um einen Fujitsu Laptop mit dem Betriebssystem Windows 10. An den Laptop wurden zwei zusätzliche Monitore angeschlossen. Auf dem Laptop wurde im Vorfeld der IPA die erforderliche Arbeitsumgebung installiert und eingerichtet.

### Datensicherung

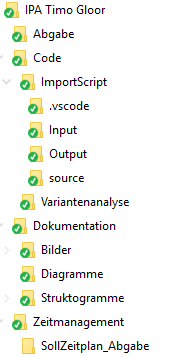


Abbildung : Ordnerstruktur

#### Dokumente

Durch die aktuelle Situation ist die Datensicherung etwas problematisch. Es steht ein Tool für die Verbindung zum Intranet der Firma zur Verfügung, jedoch ist dies durch die hohe Auslastung oftmals nicht verwendbar. Daher ist es nicht möglich, die Daten auf einem internen Share abzuspeichern.

Aus diesem Grund findet die Datensicherung auf einem lokalen Verzeichnis des Laptops statt, welches ebenfalls von Syncplicity, einem Datensynchronisationsdienst, in eine Cloud synchronisiert wird. Jeden Abend wird zusätzlich eine Kopie des gesamten Projektes erstellt und auf einer separaten SSD als Backup gespeichert. Dadurch ist es möglich, auf die Arbeitsergebnisse jedes einzelnen Tages zurückgreifen.

#### Sourcecode

Für die Versionierung des Sourcecodes war ein Repository auf GitLab.com vorgesehen. Da die Verbindung zum internen GitLab der Firma nicht gewährleistet ist, wird der Code ebenfalls lokal abgespeichert und ein tägliches Backup davon erstellt. Dies wurde so mit dem Hauptexperten besprochen.

Zusätzlich ist im Header des Codes eine History zu finden, mit der der Fortschritt der einzelnen Tage nachvollziehbar ist.

### Abteilung SI BP R&D ZG CS SAP

In der Abteilung „System Applications“ werden Funktionsblöcke für Geräte erstellt. Diese Geräte dienen alle der Gebäudeautomation. Zum Beispiel steuern diese Lüftungen, Heizkörper, Fensterläden oder ähnliche Komponenten, die es in einem intelligenten Gebäude benötigt an. Jedoch wird nicht nur die Logik dahinter entwickelt, sondern es wird auch getestet, ob der Controller wirklich das erfüllt, wie es geplant ist.

### Beteiligte Personen und Dienste

#### Personen

|  |  |
| --- | --- |
| **Rolle** | **Person** |
| **Verantwortliche Fachkraft** | Michael Speckien |
| **Hauptexperte** | Dalibor Popovic |
| **Nebenexperte** | Marcel Niederer |
| **Berufsbildner** | Martin Häusler |
| **Kandidat** | Timo Gloor |

Tabelle : Beteiligte Personen

**Organigramm**

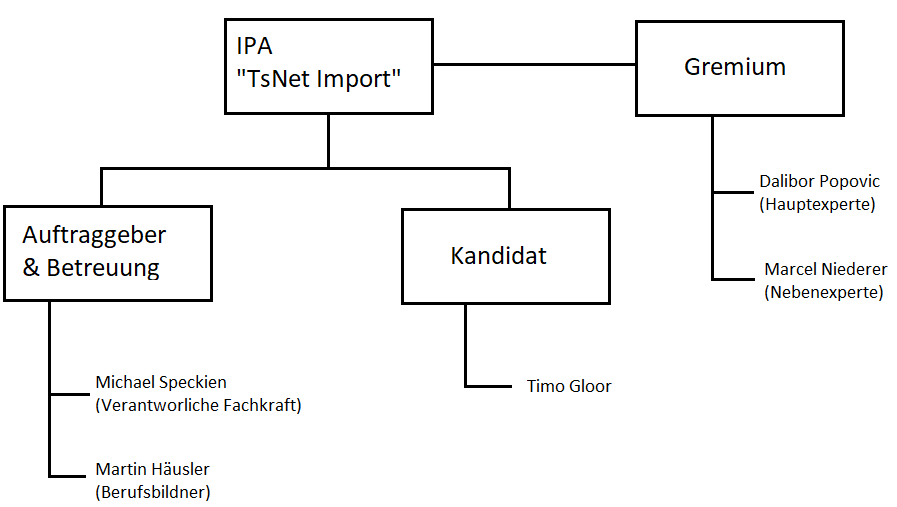


Abbildung : Organigramm

#### Dienste

|  |  |
| --- | --- |
| **Dienst** | **Verwendung** |
| Microsoft Windows 10 | Betriebssystem des Laptops |
| Microsoft Office 365 | Dokumentations- und Zeitmanagementerstellung, Mailverkehr, Einsehen und Öffnen der Excel-Testscripts |
| Circuit | Austauschmeetings |
| Visual Studio Code | Entwicklungsumgebung zum Programmieren, Einsehen von JSON-Dateien |
| Python 3.7.5 | Verwendete Programmiersprache |
| hus-Struktogrammer | Erstellung von Struktogrammen |
| draw.io | Erstellung von Diagrammen |
| Google Chrome | Informationsbeschaffung, Öffnen von PDF-Dateien |
| Synplicity | Synchronisation des Projektordners in eine Cloud |
| Snipping Tool | Erstellen von Screenshots |
| PkOrg | Verwaltung der IPA |

Tabelle : Verwendete Dienste

### Verwendete Projektmethode

Für die Umsetzung des Projektes wird die Projektmethode „IPERKA“ verwendet. Mit IPERKA wird das Projekt in sechs verschiedene Projektphasen unterteilt. Nach diesem Prinzip ist auch der Zeitplan aufgebaut und die einzelnen Tätigkeiten in die Phasen unterteilt. Diese wären:

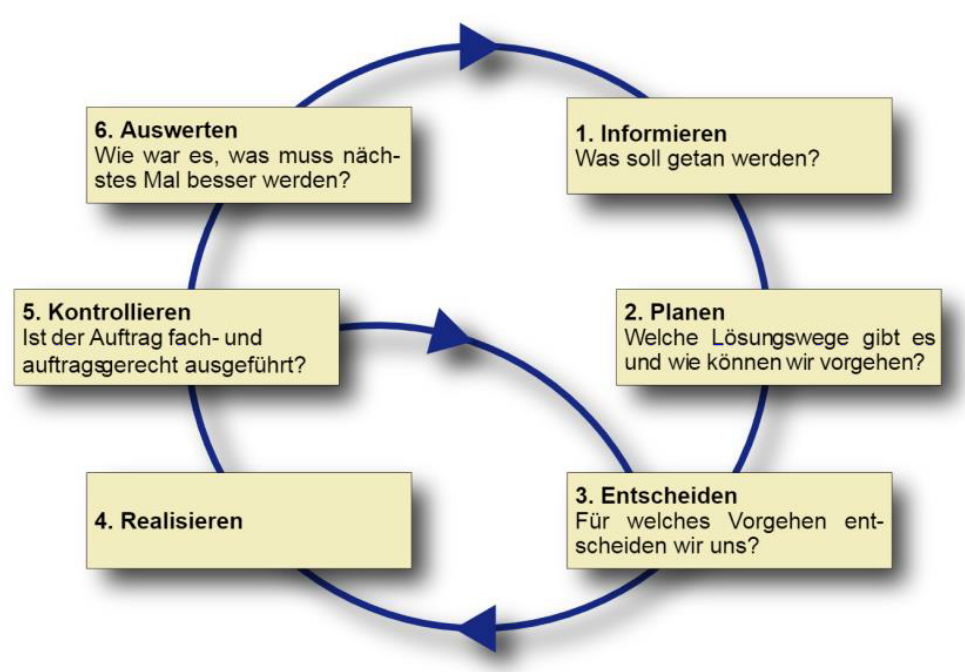


Abbildung : IPERKA

1. **I**nformieren
2. **P**lanen
3. **E**ntscheiden
4. **R**ealisieren
5. **K**ontrollieren
6. **A**uswerten

**Begründung der Wahl:**

Die Wahl für die Projektmethode IPERKA basiert auf zwei wesentlichen Punkten. Ein wichtiger Grund ist, dass es kein rein lineares Vorgehensmodell ist, sondern ebenfalls agile Aspekte mitbringt. Dies passt perfekt zu der Aufgabenstellung der IPA, da jedes Modul nach der Implementation getestet werden soll, bevor die nächsten Module entwickelt werden. IPERKA erlaubt dieses zurückspringen vom Punkt Kontrollieren zur erneuten Realisationsphase. Ausserdem wurde mit dem IPERKA-Modell bereits häufiger in der Schule gearbeitet und dort beigebracht. Dies wäre der zweite Hauptpunkt für diese Entscheidung. Des Weiteren ist das Projekt ein „Ein-Mann-Projekt“, daher würden weitere agile Methoden, wie zum Beispiel Scrum wegfallen, da sich diese eher an Teams richtet.

### Risikobeschreibung

|  |  |  |  |
| --- | --- | --- | --- |
| **Risiko** | **Beschreibung** | **Wahrscheinlichkeit** | **Auswirkungsgrad** |
| Ausfall des Laptops | Der Laptop funktioniert nicht mehr und kann nicht mehr verwendet werden. Aufgrund der Home-Office-Situation müsste mit den Experten über das weitere Vorgehen gesprochen werden. | Niedrig | Kritisch |
| Datenverlust | Da die Daten zusätzlich auf einer SSD gespeichert sind, könnte ein Backup wiederhergestellt werden | Niedrig | Mittel |
| Problemen bei der Implementation | Wenn bei der Implementation Probleme auftreten, könnte dies kritische Folgen für den Terminplan haben. Eventuell könnte das Programm dadurch nicht komplett beendet werden | Mittel | Kritisch |
| Ausfall der Internetverbindung | Durch die aktuelle Situation in der Schweiz, sind viele Leute zu Hause am Home-Office machen. Dies erzeugt eine enorme Auslastung der Internetbandbreite. In einem Worst-Case-Szenario könnte die Internetverbindung zusammenbrechen. Die Abgabe der IPA wäre somit gefährdet, wie auch Statusmeetings. Es müsste sofort ein Austausch mit den Experten über das weitere Vorgehen stattfinden. | Mittel | Kritisch |

Tabelle : Risikobeschreibung

## Planung

### Aufbau Zeitplan

Der Zeitplan wurde basierend auf dem IPERKA-Modell aufgebaut und besteht aus acht Teilen, in welche die einzelnen Tätigkeiten eingeteilt wurden.

|  |  |
| --- | --- |
| Teil 1 | Informieren |
| Teil 2 | Planen |
| Teil 3 | Entscheiden |
| Teil 4 | Realisieren |
| Teil 5 | Kontrollieren |
| Teil 6 | Auswerten |
| Teil 7 | Diverses |
| Teil 8 | Reserve |

Tabelle : Phasen des Zeitplanes

### Legende zum Zeitplan

|  |  |
| --- | --- |
| X | Soll-Zeit |
| X | Ist-Zeit |
| dd.mm.YYYY | Meilenstein |
| Offen | Tätigkeit wurde noch nicht begonnen |
| Fortlaufend | Tätigkeit wiederholt sich während des Projektes |
| In Bearbeitung | Tätigkeit ist in Bearbeitung |
| Erledigt | Tätigkeit ist abgeschlossen |

Tabelle : Legende zum Zeitplan

**Anmerkung:**

Beim Zeitplan wurden die Punkte 7.1, 7.2 und 7.3 zusammengenommen und werden als eine Tätigkeit gewertet, damit die kurzen Aktivitäten übersichtlicher sind. Am Tag macht dies ca. 30 Minuten aus. Dies wurde am Zwischengespräch mit dem Experten so besprochen.

### Soll-Zeitplan

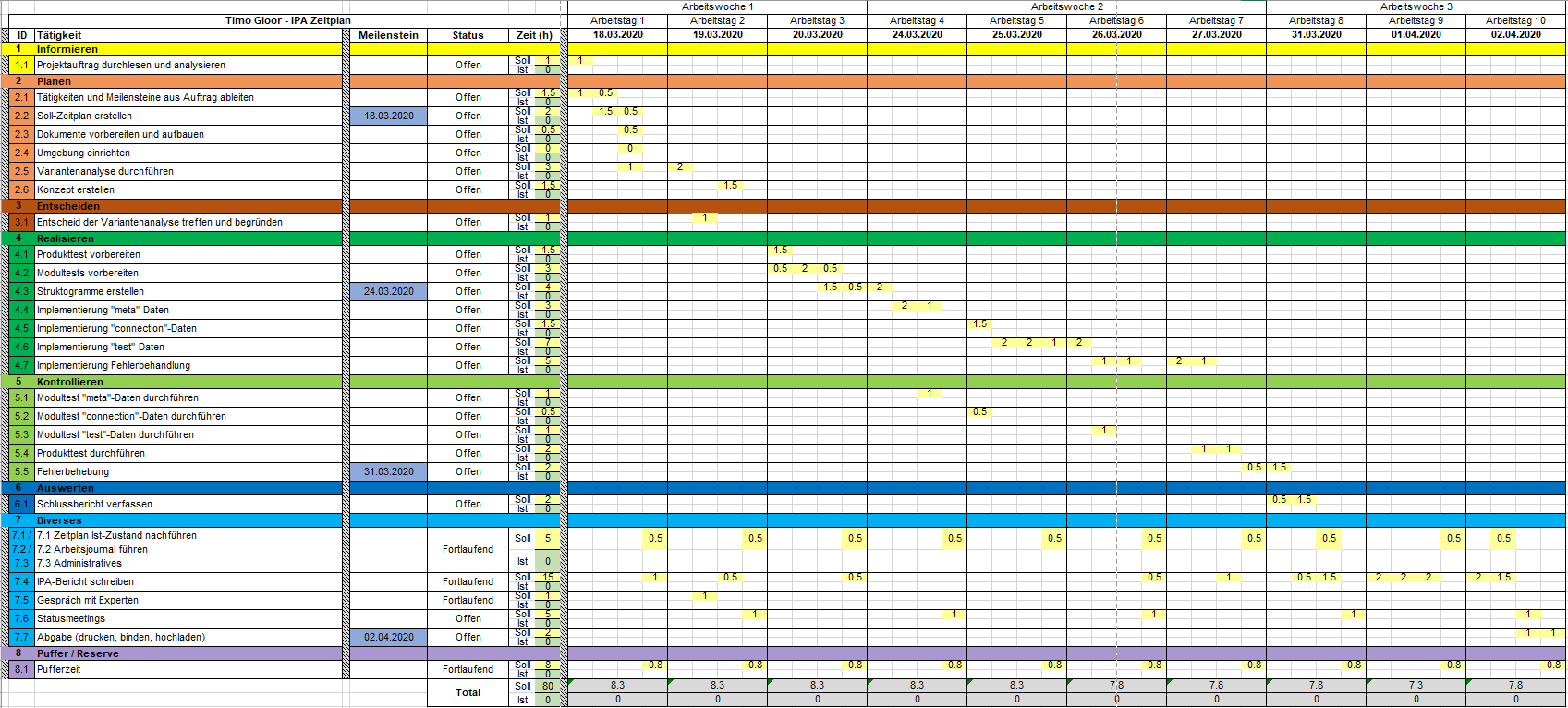


Abbildung : Soll-Zeitplan

### Ist-Zeitplan

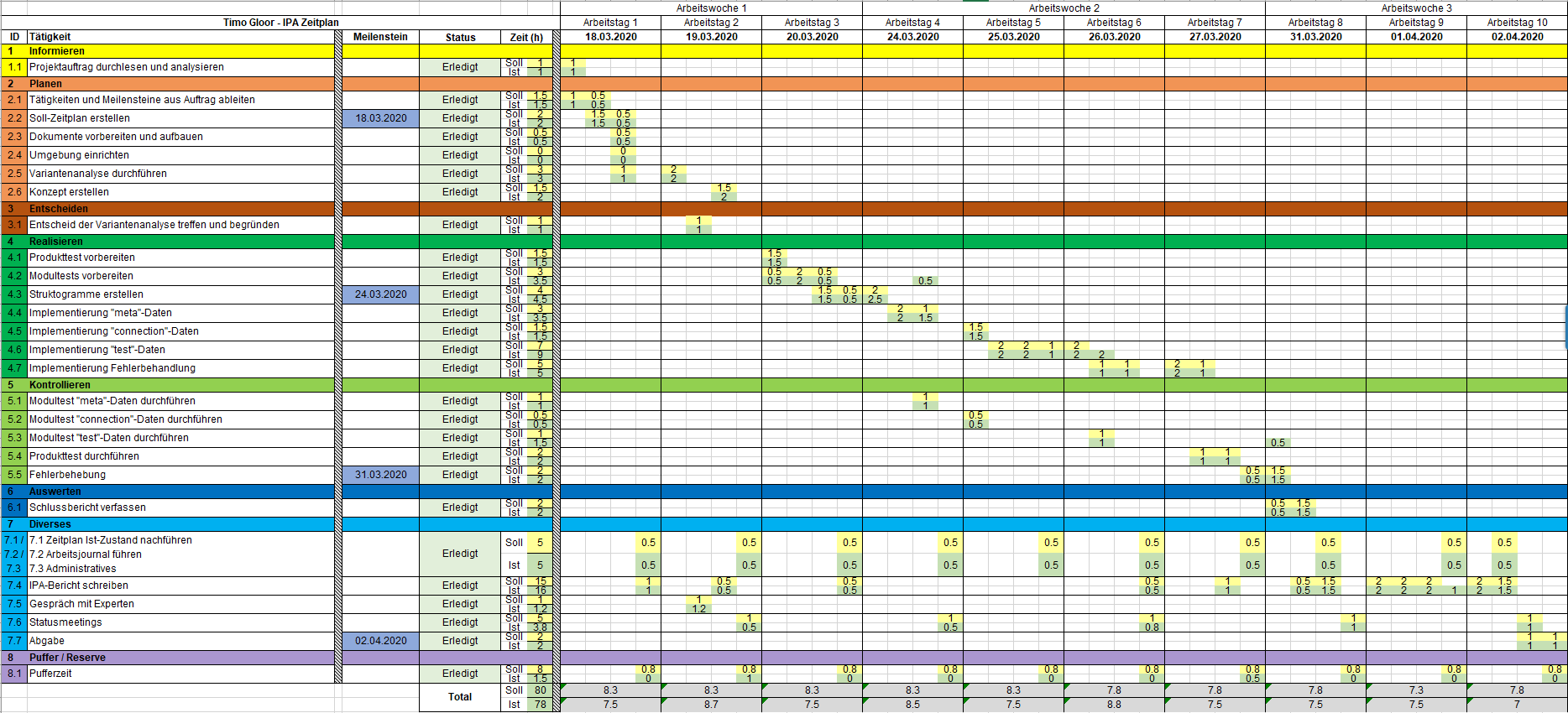


Abbildung : Ist-Zeitplan (Grüne Felder = Ist-Werte)

### Soll-Ist-Vergleich

Bei einem Vergleich der geplanten Soll-Werte mit den wirklichen Ist-Werten im Zeitplan fällt auf, dass es keine groben Abweichungen gibt. Bei ein paar Aktivitäten beträgt der Unterschied eine halbe Stunde. Dies ist allerdings nicht weiter tragisch. Lediglich für die Tätigkeit *4.6 Implementierung „test“-Daten* wurden zwei Stunden mehr benötigt als geplant.

Die totale Ist-Zeit des Projektes beträgt 78h. Also insgesamt zwei Stunden weniger als ursprünglich geplant. Dies ist vor allem auf die Pufferzeit zurück zu führen. Für jeden Tag sind 0.8h Reservezeit für unvorhersehbare Ereignisse geplant. Da von den insgesamt acht Stunden nur anderthalb Stunden gebraucht wurden, entsteht somit ein grosser Zeitunterschied. Bei der Erstellung eines Zeitplanes ist es nicht möglich, bereits im Vorfeld zu wissen, wie viel Reservezeit benötigt wird. Wenn viele technische Probleme oder sonstige Komplikationen während dem Projekt auftreten, kann es gut möglich sein, dass die Pufferzeit mit acht Stunden zu wenig ist. Allerdings kann es auch wie in diesem Fall sein, dass sie viel zu viel ist.   
Ähnlich ist es mit den Statusmeetings. Diese wurden jeweils mit einer Stunde geplant. Manchmal gingen die Meetings aber nur eine halbe Stunde oder dreiviertel Stunde. Auch dies ist im Voraus schlecht abzuschätzen.

Insgesamt gesehen, war die Fertigstellung des Projektes nie in Gefahr. Auch wenn einige Tätigkeiten etwas länger gedauert haben, war dies meistens nicht weiter schlimm, da bei anderen Aktivitäten weniger Zeit benötigt wurde

### Tätigkeiten

|  |  |  |
| --- | --- | --- |
| **ID** | **Tätigkeit** | **Beschreibung** |
| 1.1 | Projektauftrag durchlesen und analysieren | Die detaillierte Aufgabenstellung gemäss PkOrg durchlesen und verstehen, so dass der Auftrag und die Anforderungen klar sind. Nebenbei werden bereites einzelne Aufgabenfelder daraus aufgeschrieben. |
| 2.1 | Tätigkeiten und Meilensteine aus Auftrag ableiten | Den Projektauftrag in einzelne Tätigkeiten und Meilensteine aufteilen und in eine sinnvolle Reihenfolge bringen. |
| 2.2 | Soll-Zeitplan erstellen | Im Soll-Zeitplan wird der Aufwand für die einzelnen Aktivitäten und zu welchem Zeitpunkt diese erledigt werden sollen eingeplant. |
| 2.3 | Dokumente vorbereiten und aufbauen | Grobe Struktur des Dokumentes erstellen und Vorlagen für das Arbeitsjournal und das Testing erstellen. |
| 2.4 | Umgebung einrichten | Die Arbeitsumgebung gemäss den Vorgaben einrichten. |
| 2.5 | Variantenanalyse durchführen | Die beiden beschriebenen Varianten werden überprüft und untersucht. |
| 2.6 | Konzept erstellen | Nach dem eine Variante gewählt wurde, wird für das Programm ein Konzept erstellt |
| 3.1 | Entscheid der Variantenanalyse treffen und begründen | Die Varianten werden mit einander verglichen und die beste Option wird gewählt. Die Begründung wird im Bericht dokumentiert. |
| 4.1 | Produkttest vorbereiten | Vor der Implementation wird ein Produkttest mittels eines Black-Box-Testes vorbereitet. |
| 4.2 | Modultests vorbereiten | Für die einzelnen Module der Applikation wird jeweils ein Modultest in Form eines White-Box-Testes vorbereitet. |
| 4.3 | Struktogramme erstellen | Für die einzelnen Module des Programmes werden Struktogramme erstellt. |
| 4.4 | Implementierung „meta“-Daten | Modul, welches die „meta“-Daten des Testsheets ausliest und in eine JSON-Struktur bringt. |
| 4.5 | Implementierung „connection“-Daten | Modul, welches die „connection“-Daten des Testsheets ausliest und in eine JSON-Struktur bringt. |
| 4.6 | Implementierung „test“-Daten | Modul, welches die „test“-Daten des Testsheets ausliest und in eine JSON-Struktur bringt. |
| 4.7 | Implementierung Fehlerbehandlung | Implementierung der Fehlbehandlung und entsprechende Ausgaben in der Konsole. |
| 5.1 | Modultest „meta“-Daten durchführen | Den Modultest für die „meta“-Daten durchführen und dokumentieren |
| 5.2 | Modultest „connection“-Daten durchführen | Den Modultest für die „connection“-Daten durchführen und dokumentieren |
| 5.3 | Modultest „test“-Daten durchführen | Den Modultest für die „test“-Daten durchführen und dokumentieren |
| 5.4 | Produkttest durchführen | Den abschliessenden Produkttest durchführen und dokumentieren. |
| 5.5 | Fehlerbehebung | Falls Fehler aufgetreten sind, sollen diese behoben werden. |
| 6.1 | Schlussbericht verfassen | Fazit ziehen über das durchgeführte Projekt. |
| 7.1 | Zeitplan Ist-Zustand nachführen | Fortlaufende Tätigkeit: Der Zeitplan wird ständig mit den Ist-Werten der einzelnen Tätigkeiten nachgeführt. (Etwa fünf Minuten pro Tag) |
| 7.2 | Arbeitsjournal schreiben | Fortlaufende Tätigkeit: Das Arbeitsjournal wird jeweils am Ende des Tages ausgefüllt. (Ca. 20 Minuten jeweils) |
| 7.3 | Administratives | Fortlaufende Tätigkeit: Administrative Aufgaben erledigen (Täglich etwa fünf Minuten). |
| 7.4 | IPA-Bericht schreiben | Fortlaufende Tätigkeit: Die einzelnen Komponenten der IPA und deren Arbeitsschritte werden im IPA-Bericht festgehalten |
| 7.5 | Gespräch mit Experten | Am zweiten Tag der IPA (19.03.2020 um 11:00 Uhr) wird mit dem Experten und mit dem Auftraggeber ein Zwischengespräch über die IPA stattfinden. |
| 7.6 | Statusmeetings | Fortlaufende Tätigkeit: Alle zwei Tage wird mit dem Auftraggeber ein Austausch stattfinden. Diese dauern etwa eine halbe bis ganze Stunde. |
| 7.7 | Abgabe (drucken, binden, hochladen) | Abgabe der IPA. |
| 8.1 | Pufferzeit | Fortlaufende Tätigkeit: Eingeplante Reservezeit für unerwartete Ereignisse. (Jeden Tag 0.8h) |

Tabelle : Tätigkeiten

### Meilensteine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Meilenstein** | **Erledigte Arbeitsschritte** | **Erklärung** | **Datum** |
| 2.2 | Soll-Zeitplan erstellen | 1.1 bis 2.2 | Nach Erreichen dieses Punktes ist die grobe Projektplanung mit den einzelnen Tätigkeiten abgeschlossen und es kann mit dem eigentlichen Projekt begonnen werden. | 18.03.2020 |
| 4.3 | Struktogramme erstellen | 2.3 bis 4.3 | Die Planung des Programmes ist beendet und die Implementation kann beginnen. | 24.03.2020 |
| 5.5 | Fehlerbehebung | 4.4 bis 5.5 | Die Implementation ist beendet und die gefunden Fehler durch das Testing wurden behoben. Die eigentliche Aufgabe ist somit beendet. | 31.03.2020 |
| 7.7 | Abgabe | 6.1 bis 7.7 | Die IPA ist abgeschlossen und kann abgegeben werden. | 02.04.2020 |

Tabelle : Meilensteine

## Arbeitsjournal

### Zweck des Arbeitsjournales

Das Arbeitsjournal wird jeweils am Ende eines Arbeitstages geführt. Darin werden die Tätigkeiten der einzelnen Tage festgehalten. Es wird die Vorgehensweise der einzelnen Aktivitäten erklärt, über Erfolge und Misserfolge berichtet und aus der geleisteten Arbeit ein Fazit gezogen. Zum Schluss gibt es noch einen Ausblick auf die nächsten geplanten Tätigkeiten.

### Aufbau

Im Abschnitt „Tätigkeiten“ sind die Aktivitäten erwähnt, an welchen gearbeitet wurden. Der Status bezieht sich auf den aktuellen Status, wie er am Ende des Tages ist. Ebenfalls ist abzulesen, wie lange der Aufwand eigentlich geplant war und wie lange schlussendlich daran gearbeitet wurde.   
Um für den Tag ein Fazit zu ziehen, gibt es den „Reflexion“-Abschnitt. Darin wird die Vorgehensweise beschrieben und über Erfolge und Misserfolge berichtet. Unter „Nächste Schritte“ findet ein Ausblick statt, woran als nächstes gearbeitet werden sollte.

**Erwähnte Personen:**

Michael Speckien Verantwortliche Fachkraft

Dalibor Popovic Hauptexperte

Marcel Niederer Nebenexperte

### Arbeitsjournale vom 18.03.2020 bis 02.04.2020

***Arbeitsjournal Tag 1 – 18.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 1.1 | Projektauftrag durchgelesen und analysiert | Erledigt | 0 | 1 | 1 |
| 2.1 | Die Tätigkeiten und die Meilensteine aus dem Auftrag abgeleitet | Erledigt | 0 | 1.5 | 1.5 |
| 2.2 | Anhand der Tätigkeiten den Soll-Zeitplan erstellt und den Aufwand abgeschätzt | Erledigt | 0 | 2 | 2 |
| 2.3 | Grundstruktur der Dokumentation und einzelne Vorlagen erstellt | Erledigt | 0 | 0.5 | 0.5 |
| 2.4 | Umgebung war bereits eingerichtet | Erledigt | 0 | 0 | 0 |
| 2.5 | Mit der Variantenanalyse begonnen und erste Recherchen dazu gestartet | In Bearbeitung | 0 | 1 | 1 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | -0.3 | 0.2 | 0.5 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 1 | 1 |
| 8.1 | Pufferzeit wurde nicht benötigt | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Heute habe ich mit meiner IPA begonnen. Zum Start habe ich zunächst die detaillierte Aufgabenstellung, wie sie im PkOrg steht, durchgelesen und analysiert. Nachdem ich dies gemacht und ich die Aufgabe verstanden habe, habe ich die einzelnen Tätigkeiten aus dem Auftrag heraus abgeleitet und definiert. Ich habe mir dann möglichst sinnvolle Meilensteine festgelegt. Diese beiden Komponenten habe ich dann in einem Zeitplan eingetragen und den jeweiligen Aufwand abgeschätzt. Ich musste einige Male den Zeitaufwand beziehungsweise die Reihenfolge der Tätigkeiten wieder umstellen, so dass es am Ende passt. Dennoch konnte ich den Zeitplan in der geplanten Zeit erledigen und somit auch den ersten Meilenstein erreichen. Daraufhin habe ich diesen dann auch auf PkOrg hochgeladen. Anschliessend habe ich mit dem IPA-Bericht begonnen und habe schonmal eine grobe Struktur erstellt. Ebenfalls habe ich die Vorlagen für das Arbeitsjournal und für das Testing fertiggestellt, so dass ich diese dann nur noch ausfüllen muss. Da meine Umgebung bereits eingerichtet war, musste ich also nichts mehr installieren oder aufbauen. Ich konnte also direkt mit der Variantenanalyse beginnen. Dazu habe ich bereits einige Recherchen im Internet angestellt und auch schon Resultate gefunden, welche mir bei der Umsetzung später sicherlich weiterhelfen können. Danach habe ich noch ein wenig am IPA-Bericht geschrieben und zum Schluss den täglichen Arbeitsjournal-Eintrag erstellt. Für diesen habe ich etwas länger gebraucht, als ich gedacht habe. Ich hoffe, dass ich mit der Zeit weniger Aufwand dafür benötige. Generell kann ich sagen, dass ich sehr zufrieden bin mit dem ersten Tag meiner IPA. Ich hatte soweit keine grossen Probleme und auch den Zeitplan konnte ich dennoch in der geplanten Zeit erledigen. Der Zeitverlust beim Arbeitsjournal ist weiter nicht tragisch, da ich die Reservezeit nicht benötigt habe. | | | | | |
| Nächste Schritte | | | | | |
| Morgen werde ich die Variantenanalyse fertigstellen und die entsprechende Entscheidung treffen und begründen. Im Anschluss findet auch das Meeting mit dem Experten statt. Danach ist es geplant, mit meinem Konzept zur Umsetzung der Aufgabe zu beginnen. | | | | | |

***Arbeitsjournal Tag 2 – 19.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 2.5 | Variantenanalyse beendet | Erledigt | 0 | 2 | 2 |
| 2.6 | Konzept für das Programm erstellt | Erledigt | -0.5 | 1.5 | 2 |
| 3.1 | Entscheid für eine Variante getroffen | Erledigt | 0 | 1 | 1 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 1 | 1 |
| 7.5 | Zwischengespräch mit Experten geführt | Erledigt | -0.2 | 1 | 1.2 |
| 7.6 | Statusmeetings | Fortlaufend | +0.5 | 1 | 0.5 |
| 8.1 | Pufferzeit wurde verwendet | Fortlaufend | -0.2 | 0.8 | 1 |
| Reflexion | | | | | |
| Den Arbeitstag habe ich damit begonnen, indem ich meine Variantenanalyse fertiggestellt habe. Dabei konnte ich schnell ein Resultat finden und die Library auch kurz selbst testen, ob diese sich wirklich eignet. Danach konnte ich auch schon den Entscheid treffen, dass ich für die Umsetzung des Programmes die Library verwenden werde. Dies habe ich dann auch im Bericht begründet und eine kurze Anleitung zur Library geschrieben. Dann war es um 11:00 Uhr Zeit für das Zwischengespräch mit Michael Speckien und Dalibor Popovic. Leider war es aus technischen Gründen nicht möglich Herrn Niederer auch noch zur Videokonferenz hinzuzufügen. Herr Popovic ist die Checkliste durchgegangen und wir konnten noch einige Unklarheiten klären. Dann haben wir den Zeitplan noch einmal angeschaut und sind auf den Entschluss gekommen, dass ich daran noch ein paar Kleinigkeiten abändern solle. Diese habe ich dann nach dem Meeting erledigt und den Aufwand dafür auf die Reserve-Zeit genommen. Nach Fertigstellung des Zeitplanes habe ich mich wieder meiner geplanten Aufgaben gewidmet und mit dem Konzept begonnen. Das Konzept habe ich zunächst per Hand auf einem Block aufgezeichnet und habe es dann versucht mit PlantUML elektronisch umzusetzen. Dies hat mir allerdings einige Schwierigkeiten bereitet, da ich nicht herausfinden konnte, wie man bei einem Aktivitätsdiagramm eine Verbindung zu einer oberen Aktivität herstellt. Da ich nicht weitere Zeit verlieren wollte, habe ich mich dann dazu entschieden das Aktivitätsdiagramm auf der Webseite [www.draw.io](http://www.draw.io) zu gestalten. Auch wenn ich es schlussendlich umsetzen konnte, hat mir das ganze dennoch einige Zeit gekostet. Das Diagramm habe ich anschliessend in den Bericht eingefügt und das Konzept ebenfalls noch in Worten verfasst. Am Ende des Tages habe ich dann noch etwas am Bericht gearbeitet und das Arbeitsjournal geschrieben.  Insgesamt hatte ich heute einige Probleme bzw. Arbeiten, die ich nicht eingeplant hatte. Durch die Verbesserungen am Zeitplan und die Schwierigkeiten beim Erstellen der Diagramme hatte ich einen Zeitverlust. Ich hoffe, dass solche Sachen in den nächsten Tagen nicht mehr passieren werden und ich möglichst nach geplantem Aufwand arbeiten kann. | | | | | |
| Nächste Schritte | | | | | |
| Morgen werde ich mich dem Produkttest und den Modultesten zu wenden und auch bereits mit den Struktogrammen beginnen. So dass die Planungsphase bald beendet ist und ich mit der Implementation beginnen kann. | | | | | |

***Arbeitsjournal Tag 3 – 20.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 4.1 | Den Produkttest vorbereitet | Erledigt | 0 | 1.5 | 1.5 |
| 4.2 | Die Modultests vorbereitet | Erledigt | 0 | 3 | 3 |
| 4.3 | Mit der Erstellung der Struktogramme begonnen | In Bearbeitung | 0 | 2 | 2 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 0.5 | 0.5 |
| 8.1 | Pufferzeit wurde nicht verwendet | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Zum Start des Arbeitstages habe ich mit der Vorbereitung der Tests begonnen. Zunächst habe ich noch das Testkonzept verfasst und habe mich anschliessend dem Produkttest zugewandt und diesen bestmöglich vorbereitet. Als dieser beendet war, habe ich dasselbe mit den Modultests gemacht. Dabei hatte ich einige Schwierigkeiten. Es hat manchmal eine längere Zeit gedauert, bis ich wieder die Idee für einen neuen Testfall hatte. Dennoch konnte ich die Arbeitsschritte in der geplanten Zeit fertigstellen.  Auf die Vorbereitung der Tests folgte dann die Erstellung der Struktogramme für die einzelnen Module. Heute konnte ich die Struktogramme für das Meta-Modul und für das Connections-Modul erstellen. Dazu habe ich mir zuerst von Hand auf ein Blatt Notizen gemacht, wie ich den Code am besten aufbauen könnte. Nach dem ich ein paar Mal meine Ideen verworfen hatte und ich wieder von Neu beginnen konnte, habe ich es dann auch geschafft mir den Codeaufbau der Beiden Module aufzubauen. Meine Notizen von Hand habe ich dann mit dem Tool hus-Struktogrammer elektronisch umgesetzt und in meinen IPA-Bericht mit einfliessen lassen. Da dies ebenfalls perfekt in mein geplantes Zeitfenster gepasst hat, konnte ich zum Schluss noch ein wenig an der Dokumentation schreiben. Ich habe die Unterteilung der einzelnen Module nochmals erklärt und habe im Teil 1 noch das Kapitel der Projektorganisation fertig geschrieben. Dort fehlt mir jetzt nur noch ein Organigramm, welches ich dann bei der nächsten geplanten Zeit am Bericht erstellen werde. Die Reserve, bzw. die Pufferzeit habe ich heute nicht benötigt, da keine unvorhergesehenen Probleme auf mich zukamen.  Den Tag heute empfinde ich als gelungen. Ich konnte alle Tätigkeiten in der geplanten Zeit umsetzten. Auch wenn ich hin und wieder auf ein paar Probleme gestossen bin, konnte ich diese dennoch ohne weitere Folgen lösen. Ich hoffe, dass die weiteren Tage meiner IPA ebenfalls so ablaufen werden. | | | | | |
| Nächste Schritte | | | | | |
| Mein nächster IPA-Tag ist erst nächste Woche Dienstag. Dann werde ich noch das letzte Struktogramm für den Test-Bereich erstellen und somit dann auch hoffentlich den Meilenstein erreichen. Nach Abschluss dieses Meilensteines kann ich dann auch mit der Implementationsphase beginnen. Ebenfalls wird dann auch wieder ein Statusmeeting mit Michael stattfinden. | | | | | |

***Arbeitsjournal Tag 4 – 24.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 4.2 | Weitere Modultests definiert | Erledigt | -0.5 | 0 | 0.5 |
| 4.3 | Struktogramme für die Module erstellt | Erledigt | -0.5 | 2 | 2.5 |
| 4.4 | Implementation des meta-Daten-Modul durchgeführt | Erledigt | -0.5 | 3 | 3.5 |
| 5.1 | Modultest für die meta-Daten durchgeführt | Erledigt | 0 | 1 | 1 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 0.5 | 0.5 |
| 7.6 | Statusmeeting mit Michael geführt | Fortlaufend | +0.5 | 1 | 0.5 |
| 8.1 | Pufferzeit wurde nicht verwendet | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Heute Morgen habe ich noch das letzte Struktogramm fertig gestellt. Dies war für das test-Daten-Modul und mit Abstand am kompliziertesten. Bei dem Modul müssen viele Regeln beachtet werden. Ich habe es zunächst versucht sehr detailliert darzustellen. Jedoch wurde es sehr schnell verwirrend und unverständlich. Daher habe ich mich dazu entschieden, in dem Struktogramm nur die groben Anweisungen darzustellen. Eine genauere Erklärung der einzelnen Anweisungen sind dann im Code als Kommentar zu finden. Ebenfalls habe ich nebst den Struktogrammen, die einzelnen Modulen noch im Fliesstext erklärt. Insgesamt habe ich für das Erstellen der Struktogramme eine halbe Stunde mehr in Anspruch nehmen müssen als eigentlich eingeplant war. Jedoch konnte ich dann dennoch mit Fertigstellung der Struktogramme auch meinen nächsten Meilenstein erreichen. Somit ist die Planungsphase abgeschlossen und die Implementierungsphase konnte beginnen. Dazu habe ich zunächst das File erstellt und die grobe Struktur des Codes programmiert. Anschliessend begann die Umsetzung des Meta-Daten-Modules. Ich hatte einige Schwierigkeiten bei der Realisierung, bis ich schlussendlich auf ein zufriedenstellendes Resultat kam. Als Hilfestellung für die Umsetzung habe ich mich vor allem auf <https://www.w3schools.com/python/> und auf <https://www.w3schools.com/python/python_dictionaries.asp> bezogen. Auch hier habe ich eine halbe Stunde mehr benötigt als geplant war. Jedoch funktioniert das Modul nun einwandfrei. Während der Implementierung sind mir weitere Ideen für den Modultest gekommen, welche ich ebenfalls abdecken sollte. Diese habe ich dann noch nachträglich hinzugefügt. Anschliessend konnte ich die Modultests für das Meta-Daten-Modul durchführen. Bei allen Tests kam auch das erwartete Resultat heraus und konnte dies dementsprechend auch so dokumentieren. Zwischendurch hatte ich mit Michael noch ein kurzes Statusmeeting, bei dem ich ein paar Fragen klären konnte, welche mir im Verlaufe der IPA noch aufgetreten sind. Ganz am Ende des Tages habe ich dann wie schon in den Tagen davor noch das Arbeitsjournal geschrieben. Heute hatte ich wieder einige Schwierigkeiten bei den Tätigkeiten, wodurch ich auch etwas Zeit verloren habe. Da ich meine Pufferzeit aber nicht gebraucht habe, ist dies nicht weiter tragisch. Ich denke in den kommenden Tagen sollten nicht mehr weitere grosse Probleme auftreten, da ich nun bereits mit der Library gearbeitet habe und auch weiss, wie ich an welche Daten kommen kann. | | | | | |
| Nächste Schritte | | | | | |
| Für morgen ist die Implementierung des Connections-Daten-Modules geplant und die dazugehörigen Modultests. Dann werde ich mit dem Haupt-Modul bezüglich der Test-Daten beginnen können. | | | | | |

***Arbeitsjournal Tag 5 – 25.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 4.5 | Das Connections-Daten-Modul implementiert | Erledigt | 0 | 1.5 | 1.5 |
| 4.6 | Mit der Implementierung des Test-Daten-Modules begonnen | In Bearbeitung | 0 | 5 | 5 |
| 5.2 | Modultest für das Connections-Daten-Modul durchgeführt | Erledigt | 0 | 0.5 | 0.5 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 8.1 | Pufferzeit wurde nicht benötigt | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Zu Beginn des Tages habe ich das Connections-Daten-Modul implementiert. Nach kurzer Zeit kam ich schon auf eine Lösung, welche direkt auch das erwartete Resultat lieferte. Anschliessend habe ich den Code noch ein wenig «aufgeräumt» und habe alles kommentiert. Ich war froh, dass ich das Modul ohne Probleme erstellen konnte. Dies gab mir auch einen positiven Schwung für die nächsten Tätigkeiten, welche folgen sollten.  Wie nach jedem beendeten Modul, habe ich dann auch den Modultest für das Connections-Daten-Modul durchgeführt. Alle Testfälle haben das erwartete Resultat geliefert und hatte somit auch hier keine Schwierigkeiten.  Nach dem Testing konnte ich mich dann auch wieder der Implementation zuwidmen. Das Test-Daten-Modul war nun an der Reihe. Ich hatte bereits nach wenigen Minuten schon erste Probleme. Das Auslesen der Testcase-Namen aus der Overview-Tabelle hat mir extreme Schwierigkeiten bereitet, da es Probleme beim Lesen der Zellen gab. Als ich nach gewisser Zeit immer noch nicht wusste, weshalb es zu dem Fehler kommt, habe ich das Ganze mit Try-Catch umgeben. Ich werde dies bei der geplanten Zeit für die Fehlerbehandlung nochmals genauer analysieren. Anschlissend habe ich einen Algorithmus erstellt, welche für das Auslesen der einzelnen Teststeps gedacht ist. Ich hatte viel Mühe dabei, den Algorithmus zu erstellen, da ich den Lese-Prozess der Test-Daten sehr anspruchsvoll finde. Nach dem ich etwa eine Stunde auf der Suche nach einer Lösung war, konnte ich dann die Daten einlesen. Allerdings war die Struktur des JSON-Files danach nicht korrekt. Um dies zu fixen, habe ich dann nochmals etwa eine Stunde gebraucht. Als die Struktur und Inhalt stimmte, ging es noch darum den Prozess für jedes Sheet zu wiederholen. Dabei trafen noch kleinere Fehler auf, welche ich aber schnell lösen konnte. Ich habe dann den Code noch ein wenig verschönert und alles kommentiert. Nun fehlt mir noch die Umsetzung der verschiedenen Regeln für die zuschreibenden Objekt-Werte.  Gegen Ende des Tages war dann wie gewohnt wieder das Arbeitsjournal fällig. Die Pufferzeit habe ich nicht benötigt.  Den heutigen Tag empfand ich als sehr anstrengend. Das Entwickeln des Algorithmus hat sehr viel Konzentration und Nerven gekostet. Allerdings denke ich, dass ich somit den kompliziertesten Teil der IPA gemeistert habe. Ich bin zuversichtlich, dass ich das Programm so umsetzen kann, wie es auch aus der Aufgabenstellung gefordert ist. Auch wenn ich heute oft auf Schwierigkeiten gestossen bin, habe ich alle Tätigkeiten in der eingeplanten Zeit umgesetzt. | | | | | |
| Nächste Schritte | | | | | |
| Morgen werde ich die Implementierung des Test-Daten-Modules fortsetzten und hoffentlich auch beenden. Dabei muss ich noch vor allem die Regeln für die Werte einbauen und noch kleinere Sachen verbessern. Anschliessend folgt noch der entsprechende Modultest. | | | | | |

***Arbeitsjournal Tag 6 – 26.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 4.6 | Test-Daten-Modul fertig implementiert | Erledigt | -2 | 2 | 4 |
| 4.7 | Mit der Implementierung der Fehlerbehandlung begonnen | In Bearbeitung | 0 | 2 | 2 |
| 5.3 | Modultest für das Test-Daten-Modul durchgeführt | Erledigt | 0 | 1 | 1 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 0.5 | 0.5 |
| 7.6 | Statusmeeting mit Michael geführt | Fortlaufend | +0.2 | 1 | 0.8 |
| 8.1 | Pufferzeit wurde nicht verwendet | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Den Tag habe ich mit der Implementation für das Test-Daten-Modul begonnen. Ich habe einzelne Sachen verbessert und gewisse Codeteile in Funktionen unterteilt. Dann ging es noch darum, alle Regeln umzusetzen, wie sie in der Aufgabenstellung gefordert sind. Dabei stiess ich dann auf ein Problem. Es ist nicht möglich einen Struct ins JSON-File zu schrieben, ohne diesen in einen String umzuwandeln. Ich habe eine Zeit lang nach einer Lösung gesucht. Nach einer gewissen Zeit habe ich es dann zunächst sein lassen und mich den anderen Regeln gewidmet. Diese konnte ich ohne Probleme implementieren.  Als ich dann das Statusmeeting mit Michael hatte, habe ich ihn auf das Problem angesprochen. Er hat mir gesagt, es wäre nicht weiter dramatisch und ich dürfte die Struct- und Range-Objekte auch innerhalb eines Strings in das JSON-File schreiben. Ich habe Michael dann noch meinen aktuellen Stand gezeigt und nach dem Statusmeeting die Structs zu einem String umgebaut.  Ich habe anschliessend noch weiter an dem Test-Daten-Modul gearbeitet und immer wieder kleinere Sachen angepasst. Auf Probleme bin ich nicht mehr gestossen. Dennoch habe ich insgesamt zwei Stunden länger gebraucht als ich eigentlich geplant hätte.  Nach dem ich fertig war mit der Implementation des Modules, habe ich noch den dazugehörigen Modultest durchgeführt. Dabei sind mir zwei Fehler aufgefallen: Falls ein Testcase nicht als Sheet existiert, wird dieser trotzdem erstellt und bei fehlenden Werten wird der Erstellungsprozess nicht abgebrochen. Diese beiden Fehler werde ich dann in der dafür vorgesehenen Zeit für die Fehlerbehebung verbessern.  Nach dem Modultest bin ich mit der Fehlerbehandlung gestartet. Dazu habe ich zwei Funktionen erstellt, welche die entsprechenden Status Benachrichtigungen ausgeben. Ich habe ebenfalls im Code schon an ein paar Stellen die Fehlerbehandlung einbauen können. Den Rest werde ich dann morgen noch fertigstellen.  Am Ende des Tages habe ich dann noch etwas im IPA-Bericht geschrieben zu den einzelnen Modulen und habe dann das Arbeitsjournal geführt.  Der heutige Tag war zum Glück nicht ganz so anstrengend wie der gestrige Tag, dennoch konnte ich diverse Sachen erledigen und alle geplanten Tätigkeiten umsetzten. Auch wenn ich heute ca. eine Stunde mehr gebraucht habe als geplant, bin ich immer noch gut im Zeitplan und bin zuversichtlich, dass ich alles rechtzeitig erledigen kann. | | | | | |
| Nächste Schritte | | | | | |
| An meinem nächsten IPA-Tag werde ich die Fehlerbehandlung fertigstellen und somit auch die erste Version des Programmes. Dann wird der Produkttest durchgeführt und ausgewertet, sodass ich mit der Fehlerbehebung starten kann. | | | | | |

***Arbeitsjournal Tag 7 – 27.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 4.7 | Implementation der Fehlerbehandlung beendet | Erledigt | 0 | 3 | 3 |
| 5.4 | Produkttest durchgeführt | Erledigt |  |  |  |
| 5.5 | Mit der Fehlerbehebung gestartet | In Bearbeitung | 0 | 0.5 | 0.5 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 0.5 | 0.5 |
| 8.1 | Pufferzeit wurde benötigt | Fortlaufend | +0.3 | 0.8 | 0.5 |
| Reflexion | | | | | |
| Zum Start des Tages wollte ich mit der Implementation der Fehlerbehandlung fortfahren. Als ich das Script starten wollte, bekam ich allerdings immer eine Fehlermeldung, dass das angegebene Script nicht auffindbar sei. Ich habe alles überprüft und konnte nicht feststellen, wieso ich diesen Fehler bekam. Als ich auch bei einer Google-Suche nicht fündig wurde, habe ich einfach mal den Laptop neu gestartet und das Problem war behoben. Dies hat mich eine halbe Stunde gekostet, was ich allerdings auf die Pufferzeit nehmen konnte.  Da nun alles wieder funktionierte, konnte ich die Fehlerbehandlung fertigstellen. Dabei sind mir keine Schwierigkeiten aufgetreten und ich konnte gut daran arbeiten. Diese Tätigkeit konnte ich daher auch in der geplanten Zeit abschliessen. Somit war nun die erste Version des Programmes fertig, so dass ich den Produkttest durchführen konnte. Ich bin jeden einzelnen Test durchgegangen und konnte erfreut feststellen, dass alle Tests ein positives Resultat lieferten. Somit sind nur die Korrekturen für die beiden Testfälle im Test-Daten-Modultest erforderlich. Damit habe ich dann auch anschliessend noch kurz begonnen, konnte aber noch nicht die Fehler beheben. Dies ist dann am nächsten IPA-Tag fällig, wenn ich nochmals Zeit für die Fehlerverbesserung habe.  Ich habe dann noch ein wenig im IPA-Bericht geschrieben. Dort habe ich ein paar Texte und Abschnitte im Kapitel Realisieren verbessert und überarbeitet.  Am Schluss habe ich dann noch das Arbeitsjournal geschrieben.  Ich konnte heute sehr produktiv arbeiten. Abgesehen von dem technischen Problem am Morgen hatte ich heute überhaupt keine Schwierigkeiten, so dass ich gut voran gekommen bin im Projekt. | | | | | |
| Nächste Schritte | | | | | |
| An Tag 8 der IPA werde ich dann noch die Fehlerverbesserung für den fehlgeschlagenen Modultest beenden. Dies wird dann in Nachtests erneut überprüft. Wenn diese ein positives Resultat erzielen, gilt die Implementierungs-Phase als beendet und der nächste Meilenstein wäre erreicht. | | | | | |

***Arbeitsjournal Tag 8 – 31.03.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 5.3 | Nachtests für das Test-Daten-Modul durchgeführt | Erledigt | -0.5 | 0 | 0.5 |
| 5.5 | Alle Fehler behoben und das Programm fertiggestellt | Erledigt | 0 | 1.5 | 1.5 |
| 6.1 | Schlussbericht verfasst | Erledigt | 0 | 2 | 2 |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | 0 | 2 | 2 |
| 7.6 | Statusmeeting mit Michael geführt | Fortlaufend | 0 | 1 | 1 |
| 8.1 | Pufferzeit wurde nicht verwendet | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Heute Morgen habe ich die Fehlerbehebung im Programmcode fortgesetzt. Ich habe die Fehler gesucht, welche die zwei Testfälle für das Test-Daten-Modul fehlschlagen lies. Nachdem ich ein wenig herumprobiert habe und einzelne Codestellen wieder umgebaut habe, konnte ich dann die Fehler beheben. Anschliessend habe ich die Nachtests durchgeführt, welche dann ein positives Resultat lieferten. Somit bin ich also mit der Implementierungs-Phase zu Ende und den nächsten Meilenstein habe ich pünktlich erreicht. Das Programm erfüllt alle Testfälle und entspricht den Erwartungen aus der Aufgabenstellung.  Anschliessend hatte ich ein Statusmeeting mit Michael. Ich habe ihm meinen aktuellen Stand der Dinge gezeigt und konnte noch ein paar Fragen bezüglich des IPA-Berichtes stellen. Er hat mir diese beantwortet und ein wenig Feedback gegeben.  Nach dem Statusmeeting habe ich am Schlusswort geschrieben. Dafür habe ich mir zunächst die bisherigen Journal-Einträge nochmals durchgelesen, damit ich das ganze Projekt bis zum jetzigen Stand nochmals Revue passieren lassen konnte. So konnte ich die einzelnen Schwierigkeiten erneut feststellen und diese entsprechend beschreiben.  Danach hatte ich Zeit an dem IPA-Bericht generell noch zu schreiben. Ich habe die IPA-Kurzfassung geschrieben und ein Kapitel bezüglich Python hinzugefügt.  Den heutigen Tag würde ich als produktiv einschätzen. Ich konnte alle Tätigkeiten erfüllen und hatte keine Probleme. Alles lief wie geplant. Im Zeitplan ist meine Gesamt-Ist-Zeit nun etwas voraus als ich es eigentlich geplant hatte. Jedoch denke ich nicht, dass dies ein grosses Problem ist, da ich für jeden Tag eine Pufferzeit geplant habe, für unvorhersehbare Probleme. Wenn diese also nicht benötige, ist es ein gutes Zeichen, auch wenn ich dadurch einen grösseren Soll-Ist-Zeit-Unterschied habe.  Ich hoffe die letzten beiden Tage werden genauso gut verlaufen wie heute. Dann mache ich mir für Abgabe keine Sorgen. | | | | | |
| Nächste Schritte | | | | | |
| Morgen werde ich vor allem an dem IPA-Bericht schreiben. Ich möchte noch ein oder zwei Kapitel unter «Realisieren» hinzufügen. Ebenfalls fehlt noch die korrekte Angabe aller Quellen und die Begriffserklärungen für das Glossar. Diese Dinge versuche ich morgen zu erreichen. | | | | | |

***Arbeitsjournal Tag 9 – 01.04.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Fortlaufend | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Fortlaufend | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Fortlaufend | -1 | 6 | 7 |
| 8.1 | Pufferzeit wurde nicht verwendet | Fortlaufend | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Am vorletzten Tag meiner IPA habe ich heute vor allem an dem IPA-Bericht geschrieben. Ich habe unter dem Kapitel «Realiseren» noch ein weiteres Unterkapitel hinzugefügt. Des Weiteren habe ich im Teil 2 generell noch ein paar Textstellen überarbeitet und neu verfasst. Anschliessend habe ich noch im Glossar die Beschreibung für alle Begriffe hinzugefügt. Beim Quellenverzeichnis habe ich aus den Links, welche ich mir über die gesamte Dauer der IPA notiert hatte, noch die korrekten Angaben gemacht und alphabetisch sortiert.  Somit bin ich vom Inhalt her fast am Ende des Berichtes. Nun fehlt mir noch der Journal Eintrag für den letzten Tag und der Soll-Ist-Zeitvergleich, welchen ich ebenfalls morgen erstellen kann.  Ich habe für jede Tabelle und jede Abbildung in meiner Dokumentation eine Beschriftung eingefügt und anschliessend die entsprechenden Verzeichnisse am Anfang der Arbeit generieren lassen.  Am Schluss des Tages habe ich dann noch das Arbeitsjournal verfasst.  Ich konnte heute sehr produktiv arbeiten. Das Verfassen der Texte stelle mir keine grossen Schwierigkeiten. Ich habe anstatt den geplanten sechs Stunden für den IPA-Bericht sieben Stunden daran gearbeitet. Der Grund dafür ist, dass ich gerade im «Arbeitsflow» war und ich unbedingt heute den Bericht vom Inhalt her fertigstellen wollte. | | | | | |
| Nächste Schritte | | | | | |
| Morgen wird der letzte Tag meiner IPA sein. Ich muss dann noch den Soll-Ist-Vergleich erstellen und das tägliche Arbeitsjournal verfassen. Dann bin ich fertig mit dem Inhalt des IPA-Berichtes. Anschliessend sollte ich noch Zeit haben, um die Arbeit auf Grammatik und Rechtschreibung durchzulesen. Gegen Ende des Tages kommt dann die Abgabe meiner Arbeit. | | | | | |

***Arbeitsjournal Tag 10 – 02.04.2020***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tätigkeiten | | | | | |
| ID | Tätigkeiten | Status | Zeitmanagement | Soll  (h) | Ist  (h) |
| 7.1 | Beim Zeitplan die Ist-Werte nachgetragen | Erledigt | 0 | 0.1 | 0.1 |
| 7.2 | Arbeitsjournal geschrieben | Erledigt | 0 | 0.3 | 0.3 |
| 7.3 | Administratives (Mailverkehr) | Erledigt | 0 | 0.1 | 0.1 |
| 7.4 | Am IPA-Bericht geschrieben | Erledigt | 0 | 3.5 | 3.5 |
| 7.6 | Statusmeeting mit Michael geführt | Erledigt | 0 | 1 | 1 |
| 7.7 | IPA hochgeladen und abgegeben | Erledigt | 0 | 2 | 2 |
| 8.1 | Pufferzeit wurde nicht verwendet | Erledigt | +0.8 | 0.8 | 0 |
| Reflexion | | | | | |
| Heute war der letzte Tag meiner IPA. Ich habe hauptsächlich noch meinen IPA-Bericht fertiggestellt. Ich habe den Soll-Ist-Vergleich durchgeführt und die Formatierung nochmals angepasst. Zum Schluss habe ich das gesamte Dokument auf korrekte Rechtschreibung und Grammatik überprüft. Mit diesem Arbeitsjournal-Eintrag habe ich auch noch die letzte Komponente, die mir für den fertigen IPA-Bericht gefehlt hat.  Zwischendurch hatte ich mit Michael noch ein Statusmeeting. Dort haben wir kurz ein paar Sachen besprochen bezüglich der Abgabe und den weiteren Schritten.  Nun da ich alles fertig habe, werde ich meine Arbeit zusammen mit den Anhängen auf PkOrg hochladen.  Auch heute konnte ich produktiv arbeiten und die letzten paar Schritte erfolgreich beenden. Ich bin nun erleichtert, dass ich meine IPA hinter mir habe. | | | | | |
| Nächste Schritte | | | | | |
| Als nächstes kommt die IPA Präsentation mit dem Fachgespräch auf mich zu. Dafür gilt es sich nun vorzubereiten. | | | | | |

# Teil 2

## IPA Kurzfassung

### Ausgangssituation

Zum Testen von Controllern in der Gebäudeautomation werden Testscripts verwendet, welche auf Excel-Sheets basieren. In der Zukunft soll diese Testumgebung allerdings verändert werden und es wird an einem neuen Tool, nämlich das «TsOpen»-Tool gearbeitet. Damit die alten bereits erstellten Tests dennoch verwenden werden können, wird eine Import-Funktion benötigt. Das Import-Script wandelt die einzelnen Testcases innerhalb eines Excel-Test-Files in .json-Dateien um. Somit kann das TsOpen-Tool auf die .json-Files zugreifen und den Test einlesen, damit dieser weiterverwendet werden kann. Die Umwandlung ins JSON-Format ermöglicht ebenfalls das Ablösen von Excel. Dies liefert den Vorteil, dass keine Office-Lizenzen mehr für das Testing von Controllern benötigt wird

### Umsetzung

Im Rahmen der IPA soll nun dieses Import-Script erstellt werden. Für das Projekt standen zehn Arbeitstage (80h) zur Verfügung. Das Projekt wurde mittels der Projektmanagementmethode IPERKA abgewickelt. Damit ein ständiger Soll- und Ist-Vergleich möglich ist, wurde zunächst ein Zeitplan erstellt, in dem alle Tätigkeiten des Projektes mit dem abgeschätzten Aufwand eingetragen sind. Die Tätigkeiten wurden in die einzelnen Projektphasen unterteilt.

Anschliessend begann die Planungsphase, in der zunächst eine Variantenanalyse durchgeführt wurde. Anhand der gewählten Variante konnte dann das Konzept für das Programm entwickelt werden. Nach dem das Konzept entwickelt wurde, konnte daraus das Programm in verschiedene Module abgegrenzt werden. Für jedes Modul wurde nach Fertigstellung ein Modultest durchgeführt. Diese sind als White-Box-Tests aufgebaut. Besteht das Modul alle Tests, kann mit der Implementierung des nächsten Modules begonnen werden. Nach Fertigstellung des Programmes soll ein Produkttest durchgeführt werden. Dieser ist ein Black-Box-Test. Alle Testfälle wurden vor Beginn der Implementation definiert.

Nach dem die Struktogramme erstellt wurden und somit der grobe Ablauf der Module geplant war, konnte mit der Implementierung des Scripts begonnen werden. Nach dem die Hauptmodule und die dazugehörigen Tests beendet wurden, wurde noch die Fehlerbehandlung implementiert. Anschliessend wurde mit dem Produkttest die gesamte Funktionalität überprüft.

Fehler, die während dem Testing aufgetreten sind, wurden gekennzeichnet und behoben. Entsprechende Nachtests wurden dann erneut durchgeführt, um sicher zu gehen, dass nun die Anforderung des Tests erfüllt wird.

### Ergebnis

Nach Abschluss der IPA ist eine Python-Applikation vorhanden, welche es ermöglicht alte Excel-Testsheets und deren Testcases in ein JSON-Format zu bringen und diese an der gewünschten Stelle abzuspeichern. Dabei sind alle Anforderungen, welche in der Aufgabenstellung gefordert werden, abgedeckt. Der Code ist kommentiert und im zweiten Teil des IPA-Berichts dokumentiert.

## Entscheiden

### Variantenanalyse

#### Beschreibung

Für das Projekt soll eine Variantenanalyse durchgeführt werden. Dabei sollen Optionen geprüft werden, wie die Daten aus dem Excel-File eingelesen und verarbeitet werden können. Die optimale Lösung dafür wäre eine Python-Library, mit der der Zugriff auf die Excel-Daten funktioniert. Falls dies jedoch nicht möglich ist, müsste das Excel-File zuerst in ein anderes Format, bspw.: CSV, HTML oder XML gebracht werden

#### Vorgehen

Da die optimale Variante, die Lösung mittels einer Python-Library zum Einlesen der Excel-Daten wäre, wird der Fokus vor allem auf der Suche nach solch einer Library liegen. Dafür wird mittels des Internets eine Recherche stattfinden, ob es solch eine Library gibt und ob es sich eignet, diese im Programm zu verwenden.

#### Resultat der Recherche

Die Internet-Recherche ergab, dass es diverse Python-Libraries gibt, welche für den Zugriff auf Excel-Dateien verwendet werden können. Da allerdings nicht jede dieser Libraries die alte Excel-Version(.xls) unterstützt, fallen einige Bibliotheken wieder weg. Dennoch gibt es Möglichkeiten auch ältere Exceldateien einzulesen. Insbesondere scheint die Library «xlrd» optimal für das Projekt zu sein. Auf der Webseite <https://xlrd.readthedocs.io/en/latest/> ist die Bibliothek ebenfalls dokumentiert. Es benötigt kein grosses Installationsverfahren und die Funktionen für den Datenzugriff erweisen sich als simpel. Bereits nach wenigen Zeilen Code ist es möglich, auf gewünschte Zellen-Inhalte eines bestimmten Excel-Sheets zu zugreifen.

#### Entscheid

Die Wahl der Variante fällt sehr leicht. Das optimale Verfahren wäre der Datenzugriff mittels einer Python-Library, welche Exceldateien öffnen und lesen kann. Die Recherche ergab, dass genau dies mit der Bibliothek «xlrd» möglich ist. Ebenfalls erzeugt die Library keinen grossen Aufwand für die Anwendung, was die Sache nochmals vereinfacht. Es wäre also ein unnötiger Zwischenschritt, wenn das Excel-File erst in ein anderes Format umgewandelt werden würde, wenn solch eine Library zur Verfügung steht. Daher lautet der Entscheid der Variantenanalyse, dass das Programm mit der Python-Library «xlrd» umgesetzt wird.

Eine genau Anleitung der projektrelevanten Funktionalitäten der Library ist im Kapitel «2.3.5 Excel-Daten Zugriff mittels Python-Library» zu finden.

## Realisieren

### TsNet Import Script

#### Zweck

Zum Testen von Controllern in der Gebäudeautomation werden Testscripts verwendet. Diese wurden bis zum jetzigen Zeitpunkt mit dem auf Excel basierenden Tool «TsNet» erstellt. In der Zukunft soll eine neue Testumgebung geschaffen werden. Damit die alten Testfälle aber nicht verloren gehen, benötigt es eine Import-Funktion für diese alten Testscripts. Das Ziel des neuen Test-Frameworks ist es ebenfalls weg von einer Excel-Umgebung zu kommen. Daher sollen die Scripts in ein anderes Format gebracht werden. Nämlich in ein JSON-Format.

Das Import-Script liest die alten Excel-Tests (.xls) ein und wandelt die einzelnen Testfälle darin in eine neue .json-Datei um. In der neuen Testumgebung können dann die .json-Files eingelesen werden und so aufbereitet werden, dass sie weiterhin verwendet werden können.

#### Anwendung

Bevor das Script gestartet werden kann, muss sowohl Python wie auch die Library «xlrd» auf dem System installiert sein. Die entsprechenden Anleitungen dazu sind im Kapitel «2.3.4 Python» und unter «2.3.5 Excel-Daten Zugriff mittels Python-Library» zu finden.

Die Bedingung der Applikation erfolgt via einer Konsole. Beispielsweise das Standard Windows-Terminal. Das Import-Script verlangt zwei Parameter: einmal den Pfad für das Excel-Inputfile und als zweiten Wert den Pfad zum Ausgabeverzeichnis, in welchem die JSON-Files erzeugt werden sollen.

Das Script lässt sich in der Konsole auf folgende Art und Weise starten:

*python tsNetImport.py SourceFilepath DestinationFolder*

Dabei ist es wichtig, dass die Konsole im gleichen Verzeichnis wie das Python-Script ist, ansonsten müsste der Pfad für das Script ebenfalls angegeben werden.

Die beiden Parameter sollten innerhalb von Anführungs- und Endzeichen übergeben werden, da ein Leerschlag der Trennung der Parameter dient. Würde im Pfad ein Leerschlag vorkommen, würde dies den Parameter aufteilen. Innerhalb von " " kann dies nicht passieren.

**Beispiel eines korrekten Aufrufes:**

*python tsNetImport.py "C:\tmp\TsNetV1\_Example.xls" "C:\tmp\Output"*

Das Aufrufen des Scripts ist gleichzeitig auch schon die einzige User-Interaktion. Das Programm wird in der Konsole über eventuelle Fehler, Warnungen oder sonstige Statusmeldungen informieren. Zum Schluss erfolgt eine zusammenfassende Ausgabe über den Erfolg des Import-Prozesses.

### Konzept

#### Aktivitätsdiagramm

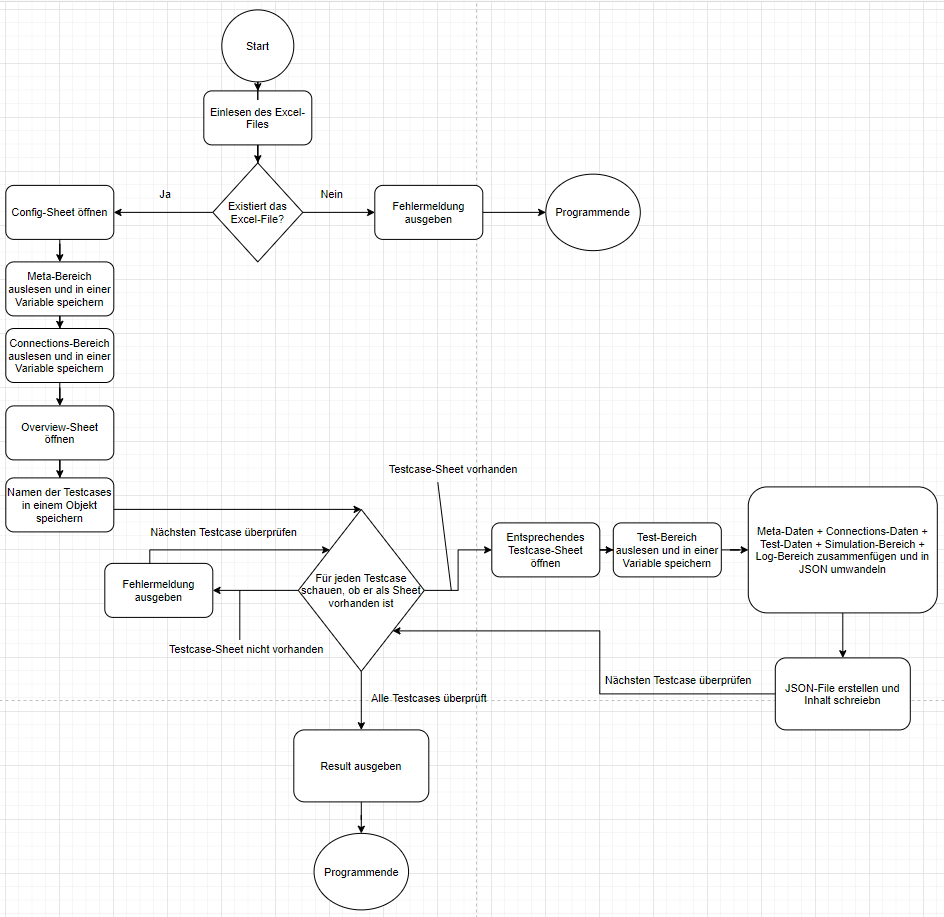


Abbildung : Aktivitätsdiagramm

Für das Veranschaulichen des Konzeptes wurde ein Aktivitätsdiagramm gewählt, da sich dieses für eine Applikation, welche nicht in diverse Klassen beziehungsweise Objekte unterteilt ist, am besten anbietet.

#### Genereller Aufbau der Applikation

Wie oberhalb erwähnt ist das Programm nicht in Klassen oder Objekte unterteilt. Der Grund dafür ist, dass das Script einen prozeduralen Ablauf hat; Ein Modul ist vom nächsten Modul gefolgt. Es würde sich nicht lohnen, dies in Klassen zu unterteilen, da die Module auf dieselben globalen Objekte zugreifen müssen.

Am Anfang des Scripts werden die übergebenen Paramater eingelesen und es wird überprüft, ob das angegebene Excel-File existiert. Falls dies nicht der Fall ist, wird eine Fehlermeldung ausgegeben und das Programm wird beendet.

Wenn das File gefunden wurde, soll darauf zugegriffen werden. Zunächst wird das Config-Sheet geöffnet und die Meta-Daten ausgelesen werden. Diese werden dann in einer Variable zwischen gespeichert. Dasselbe passiert mit den Connections-Daten, auch diese werden abgespeichert. Der Grund für das Zwischenspeichern dieser beiden Bereiche ist, da in jedem Testcase, der generiert wird, diese Abschnitte identisch sind, bis auf zwei Werte, welche allerdings zum Schluss des Scripts individuell geändert werden. So kann später alles zusammengeführt werden und ins JSON-File geschrieben werden.

Nachdem die wesentlichen Daten aus dem Config-Sheet entnommen wurden, geht es weiter mit dem Overview-Sheet. Darin befinden sich alle Testcase-Namen. Diese werden in einem Objekt abgespeichert, so dass danach durch die einzelnen Testcases durchiteriert werden kann.

Es folgt die Schleife, bei der jeder Testcase-Name überprüft wird, ob er auch als Sheet im Excelfile vorhanden ist. Ist ein Testcase-Sheet vorhanden, wird dies geöffnet. Nun werden darin die einzelnen Daten und Tests herausgelesen und in einer temporären Variable abgespeichert. Anschliessend werden die zuvor abgespeicherten Meta- und Connection-Bereiche genommen und zusammen mit dem Test-Bereich gemeinsam in ein JSON-Format gebracht. Zum Schluss werden noch die leeren Bereiche für das Logging und die Simulation hinzugefügt. Wenn nun alle Bereiche zusammen sind, wird das finale JSON-File erzeugt und mit dem eben erzeugten Inhalt gefüllt. Sobald das JSON-File erstellt wurde, wird eine Statusmeldung ausgegeben und die nächste Iteration der Schleife mit dem nächsten Testcase-Namen beginnt.   
Wenn alle Testcases abgearbeitet wurden, gibt das Programm eine Zusammenfassung in einer finalen Status-Ausgabe in der Konsole aus.

#### Unterteilung der Module

Die Applikation wird in drei Hauptmodule unterteilt:

* Meta-Daten
* Connections-Daten
* Test-Daten

Diese drei Bereiche bilden schlussendlich die Hauptstruktur des erforderlichen JSON-Formates. In der Implementationsphase wird zunächst das Modul für die Meta-Daten umgesetzt, darauf folgen die Connections-Daten und zum Schluss der grösste Teil mit den Test-Daten. Sobald ein Modul fertiggestellt wird, wird dieses direkt durch bereits erstellte White-Box-Tests überprüft und allenfalls verbessert. Das Ziel ist, dass sich die Module so aufeinander aufbauen können.

Gegen Ende werden alle Module genommen und so zusammengestellt, dass das Script das erwartete Resultat liefert. Dazu gehört neben diesen Modulen auch eine Fehlerbehandlung, welche über das gesamte Script zum Einsatz kommt.

Eine genauere Beschreibung der einzelnen Module ist im Kapitel 2.3.3 zu finden.

#### Fehlerbehandlung und Statusmeldungen

Da die Fehlerbehandlungen verteilt über das gesamte Script benötigt wird, kann diese nicht in separates Modul unterteilt werden.

Bei jeder kritischen Stelle im Code, bei der ein Fehler auftreten könnte, wird dieser auch abgefangen und eine entsprechende Statusmeldung ausgegeben. Für die Statusmeldungen gibt es zwei Funktionen, welche von jeder Stelle im Code aufgerufen werden können:

**PrintStatusNotification(notificationType, msg, location, nextStep)**

Beschreibung:

Diese Funktion ist dazu da, dass in der Konsole eine Statusmeldung ausgeben werden kann.

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| notificationType | Art der Meldung (ERR, WARN, INFO) | String |
| msg | Auszugebener Fehler-/Infotext | String |
| location [Optional] | Stelle, an der der Fehler aufgetreten ist (Zelle, Sheet) | String |
| nextStep [Optional] | Weiteres Vorgehen des Programmes (Continue, Exit) | String |

Returnwert:

Es handelt sich dabei um eine Void-Funktion und hat somit keinen Returnwert.

**PrintSummary(createdTestcases, expectedTestcases)**

Beschreibung:

Erstellt eine finale Zusammenfassung über den Erfolg des Import-Vorganges. Beinhaltet eine Auflistung über die erstellten Files. Beispielsweise so:

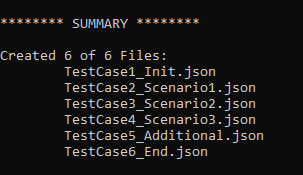


Abbildung : Beispiel Summary-Ausgabe

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| createdTestcases | Namen der erstellten JSON-Files | String-Array |
| expectedTestcases | Name aller Sheets, welche in der Overview-Tabelle aufgelistet sind | String-Array |

Returnwert:

Es handelt sich dabei um eine Void-Funktion und hat somit keinen Returnwert.

#### Helfer-Funktionen

Für einen saubereren Code gibt es einige Helfer-Funktionen. Diese sollen redundante Codestellen zusammenfassen, damit diese nur an einem Ort auftreten:

**ConvertCoordinate(coordinate)**

Beschreibung:

Diese Funktion ermöglicht es, Koordinaten von der Excel-Schreibweise in die X- und Y-Koordinaten umzuwandeln. Diese werden benötigt, um mit der Library auf die Zellen zuzugreifen. Die Funktionalität ist begrenzt auf die Spalten-Werte von A-Z.

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| coordinate | Koordinate der Excel-Zelle als Excel-Schreibweise. Zum Beispiel: «A:12», «B:1», «C:4» | String |

Returnwert:

X- und Y-Koordinate der Excel-Zelle. Beispielwerte für die oben genannten Koordinaten: [0, 11], [1, 0], [2, 3]

**CreateTestStep(funct, val)**

Beschreibung:

Diese Funktion erstellt einen neuen Testschritt anhand des ausgelesenen Wertes der Zelle und den dazugehörigen Objekt-Properties.

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| funct | Funktion des Testschrittes; «Set» oder «Verify» | String |
| val | Inhalt der Zelle | Float oder String |

Returnwert:

Gibt den erzeugten Teststep zurück oder False, falls ein Fehler aufgetreten ist.

**CheckDataType(val)**

Beschreibung:

Diese Funktion erlaubt es, den gelesenen Zellenwert auf den Datentyp zu überprüfen. Falls es notwendig ist, wird der Datentyp umgewandelt.

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| val | Inhalt der Zelle | Float oder String |

Returnwert:

Gibt den Inhalt der Zelle in passendem Datentyp zurück.

**CreateJSONFile(outputPath, sheetName, content)**

Beschreibung:

Funktion, zum Erstellen eines JSON-Files mit dem Inhalt des Tests.

Parameter:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Beschreibung** | **Typ** |
| outputPath | Pfad zum Directory, in welchem das File erzeugt werden soll | String |
| sheetName | Name des Testcase-Sheets | String |
| content | Inhalt, welcher ins File geschrieben werden soll | String |

Returnwert:

Gibt den Inhalt der Zelle in passendem Datentyp zurück.

#### Umwandlung der Excel-Daten in eine JSON-Struktur

Um die Excel-Daten in eine JSON-Struktur zu bringen, wird mit Dictionaries gearbeitet. Ein Dictionary ist eine Art Sammlung von Objekten. Diese Objekte besitzen einen Schlüssel und einen Wert (Key and Value). Im Schlüssel wird meistens der Property-Namen abgespeichert und im Werte-Bereich der dazugehörige Wert.

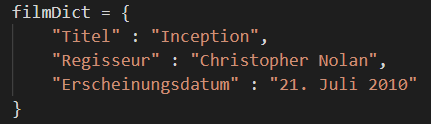


Abbildung : Beispiel Dictionary

In dem Beispiel oberhalb sind die Schlüssel jeweils auf der linken Seite erkennbar und die dazugehörigen Werte auf der rechten Seite.

🡪 Bsp. Für Objekt 1: Schlüssel: Titel, Wert: Inception

Da die JSON-Struktur dasselbe Key-and-Value-Prinzip befolgt, eignen sich die Dictionaries sehr, um die Daten zunächst zu verwalten und zwischenzuspeichern.

Im Programm gibt es drei Haupt-Dictionaries. Für jeden Bereich eines (meta-Bereich, connections-Bereich, test-Bereich). Diese beinhalten jeweils die gesamten Daten, welche für den Abschnitt schlussendlich auch ins File geschrieben werden sollen.

Es werden zunächst mittels der Library «xlrd» die Zell-Werte aus dem Excel-File ausgelesen. Die erhaltenen Werte dienen später als Value-Eintrag. Zusammen mit dem Property-Namen des Wertes, also dem Key-Eintrag, werden diese dann zum entsprechenden Dictionary hinzugefügt. Dadurch ist es möglich die ausgelesenen Werte auf einfache Art und Weise bereits in eine JSON ähnliche Struktur zu bringen.

Wenn alle Daten ausgelesen wurden, ist es möglich die verschiedenen Dictionaries zu vereinen und die Daten aus allen Bereichen miteinander in eine einzige Struktur zu bringen. Dies wird ganz am Schluss des Scripts gemacht. Alle Daten aus allen Bereichen werden zusammengebracht und anschliessend in ein JSON-File geschrieben. Dafür gibt es in der Python-Library «json» eine Funktion, um den Inhalt eines Dictionaries in ein JSON-File zu schreiben. Diese wird aufgerufen und das JSON-File mit dem gewünschten Inhalt wird erzeugt.

### Module

#### Meta-Daten

##### Struktogramm Meta-Daten

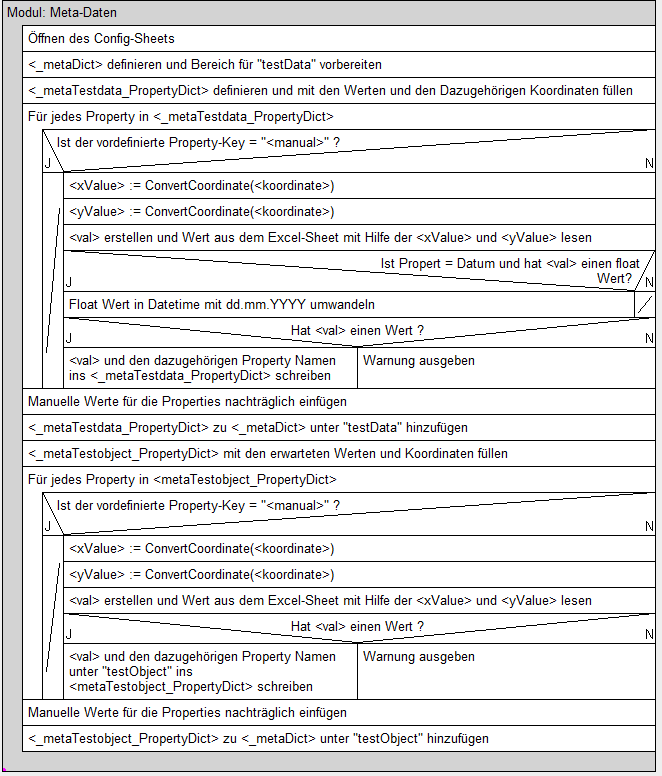


Abbildung : Struktogramm Meta-Daten-Modul

##### Erklärung Meta-Daten

Für das Meta-Daten-Modul gibt es drei verschiedene Dictionaries. Das *\_metaDict* dient zum Abspeichern des gesamten Meta-Daten-Inhaltes, sodass dies am Schluss des Scripts zu den anderen Modulen hinzugefügt werden kann und der JSON-Inhalt erzeugt werden kann.

Dann gibt es noch das *\_metaTestdata\_PropertyDict* und das *\_metaTestObject\_PropertyDict.* Diese beiden Dictionaries werden dafür verwendet, um die erwarteten Property-Namen und die dazugehörigen Koordinaten im Excel-File vorab zu definieren.

Teilweise enthalten diese anstatt der Koordinate auch den Wert «<manual>». Dies wird für Properties benötigt, welche nicht einfach aus dem Excelfile herausgelesen werden können.

Zum Lesen der Werte, wird zunächst das Config-Sheet geöffnet. Nun kann durch das erste Dictionary mit den «testData»-Daten iteriert werden. Dabei wird zunächst für jedes Property überprüft, ob es sich um eine Koordinate handelt oder es einen manuellen Wert verlangt. Wenn Koordinaten verfügbar sind, werden diese erst noch in X- und Y-Werte konvertiert und damit anschliessend der Wert der Zelle ausgelesen.

Der erhaltene Wert wird überprüft, ob es sich dabei um ein Datum handelt, falls dies der Fall ist, wird das Datum noch in das gewünschte Format (dd.mm.YYYY) konvertiert.

Hat der Wert überhaupt keinen Wert, wird eine Fehlermeldung ausgegeben. Ansonsten wird der Wert mit dem Property-Namen im Dictionary zwischengespeichert.

Wenn alle «testData»-Werte abgearbeitet wurden, wiederholt sich der Prozess für die «testObject»-Daten. Das Vorgehen ist dabei grösstenteils dasselbe wie eben erklärt. Wenn auch diese Daten eingelesen wurden, werden die beiden Bereiche ins Haupt-Dictionary *\_metaDict* reingeschrieben.

#### Connections-Daten

##### Struktogramm Connections-Daten

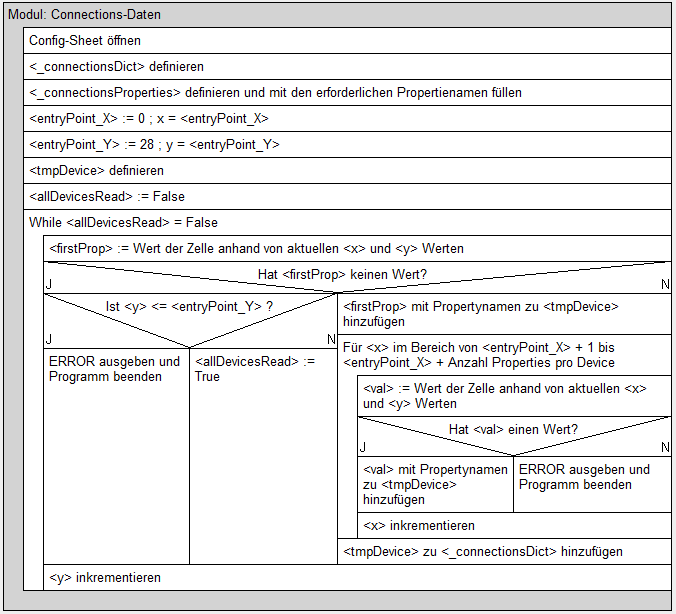


Abbildung : Struktogramm Connections-Daten-Modul

##### Erklärung Connections-Daten

Zum Auslesen der Connections-Daten wird zunächst auf das Config-Sheet gewechselt. Es folgt der gleiche Prozess wie zu Beginn des Meta-Daten-Modules: Es wird ein Dictionary erzeugt, in welchem die Connections-Daten später abgespeichert werden können. Bei diesem werden die Property-Namen allerdings bereits definiert.

Mit zwei Variablen werden die Einstiegspunkte für die X- und Y-Koordinate im Excel-File definiert. Mit den Einstiegsvariablen wird dann durch jede Reihe für den Connections-Bereich iteriert, bis kein Device mehr vorhanden ist. Dies merkt das Programm daran, dass das erste Property einer neuen Reihe keinen Wert mehr besitzt. Ist ein Wert vorhanden, wird dieser mit dem Property-Namen ins Dictionary geschrieben. Anschliessend werden alle anderen Werte für das Device ausgelesen und ebenfalls abgespeichert. Sobald alle Zellen eingelesen wurden für das aktuelle Device, werden die Daten ins *\_connectionsDict* geschrieben und abgespeichert.

#### Test-Daten

##### Struktogramm Test-Daten

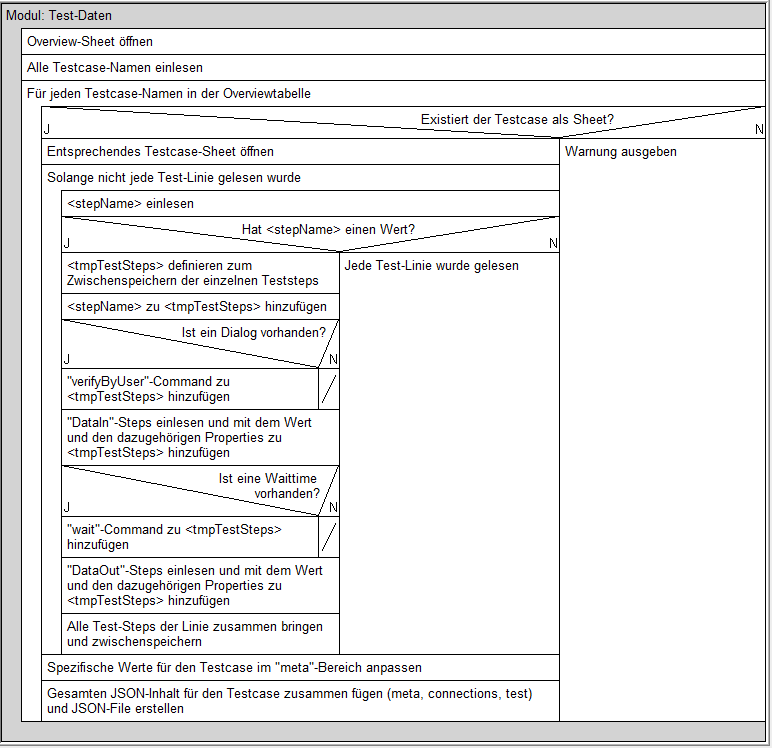


Abbildung : Struktogramm Test-Daten-Modul

##### Erklärung Test-Daten

Zunächst wird das Overview-Sheet geöffnet, bevor die einzelnen Testcases abgearbeitet werden können. Im Overview-Sheet gibt es eine Übersicht mit den einzelnen Testcases und wie diese als Excel-Sheet heissen. All diese Namen werden ausgelesen und in einem Array abgespeichert. Durch dieses erzeugte Array wird anschliessend auch iteriert.

Zu Beginn der Schleife wird zunächst überprüft, ob das entsprechende Testcase-Sheet überhaupt in der Excel-Mappe existiert. Ist dies nicht der Fall wird eine entsprechende Warnung ausgegeben und die nächste Iteration der Schleife mit dem nächsten Testcase beginnt. Wenn das Sheet vorhanden ist, wird dieses geöffnet.

Es folgt eine weitere Schleife, mit der durch die einzelnen Steps der Test-Linie durchgegangen werden kann. Zunächst wird der «stepName» ausgelesen und überprüft, ob dieser einen Wert besitzt. Falls nicht, wird eine Fehlermeldung ausgegeben und den Prozess für das aktuelle Testsheet abgebrochen.

Hat der «stepName» einen Wert, wird dieser in einem temporären Dictionary abgespeichert. Wenn für den Test ein Wert im Dialog vorhanden ist, wird dieser zu einem «verifyByUser»-Command umgewandelt und entsprechend im Dictionary abgespeichert. Nun werden alle Actions des Tests abgearbeitet und gemeinsam mit den Werten der Objekt-Properties zum temporären Dictionary hinzugefügt. Wenn zwischen den Action-Schritten und den Verifikations-Schritten noch eine Wartezeit benötigt wird, wird diese mit dem Wert in der «Waitingtime»-Zelle angegeben. Ist ein Wert vorhanden, so wird dieser zu einem «wait»-Command umgebaut und zu den restlichen Steps hinzugefügt.

Für die «DataOut»-Werte kommt dasselbe Vorgehen zum Einsatz, wie es bereits für die «DataIn»-Werte geschehen ist. Es wird immer der erwartete Wert genommen und mit den Objekt-Properties zusammen zu einem Teststep gebaut und dann im temporären Dictionary abgespeichert. Ist der Test fertig, wird diese temporäre Objekt genommen und zum Haupt-Dictionary mit allen weiteren Test-Linien des Tescases hinzugefügt. Anschliessend beginnt der nächste Durchlauf der Schleife und die nächste Test-Linie wird generiert. Dies solange, bis alle Linien abgearbeitet wurden.

Dann werden die einzelnen Test-Linien zusammengefügt und mit dem Meta- und Connections-Bereich zusammen in eine JSON-Struktur vereint. Das JSON-File wird erstellt und der eben erzeugte Inhalt wird hineingeschrieben.

#### Einsatz der Module im Code

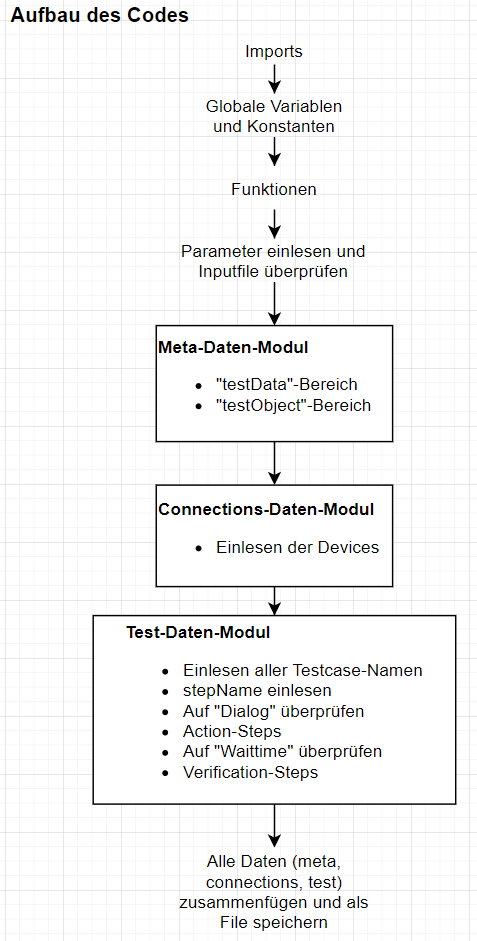


Abbildung : Codeaufbau

Das Diagramm oberhalb soll veranschaulichen, wie und an welcher Stelle die Module im Code vorkommen und wie der Code generell aufgebaut ist.

### Python

Das Script wurde mittels der Programmiersprache Python entwickelt. Die verwendete Version ist Python 3.7.5.

**Python installieren**

Unter <https://www.python.org/downloads/> kann eine gewünschte Python Version heruntergeladen werden. Nach dem Auswählen der Version, kann eine passende Installationsdatei gewählt werden. Dabei sind die «executable installer»-Dateien zu empfehlen. Nach dem Download ist die Installationsdatei ausführbar. Es öffnet sich ein Fenster, bei dem es wichtig ist, die Checkbox für «Add Python to PATH» anzuwählen. Danach kann die Installation via «Install Now» gestartet werden.

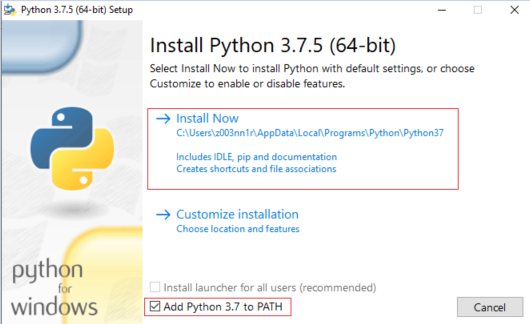


Abbildung : Python Installation

Um zu prüfen, ob die Installation erfolgreich war, kann dies in einem Terminal, beispielsweise das Windows-CMD, mit dem Befehl python getestet werden. Es sollte eine Ausgabe wie folgt ausgegeben werden:

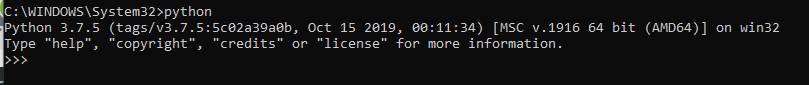


Abbildung : Python Installation überprüfen

### Excel-Daten Zugriff mittels Python-Library

#### Vorbereitung

Der Zugriff auf die Excel-Daten erfolgt mittels der Library «xlrd». Diese Bibliothek ermöglicht es, auf Excel-Files zuzugreifen und daraus Werte einzelner Zellen auszulesen.  
Damit das Ganze funktioniert, muss die Library zunächst installieret werden. Dies erfolgt sehr einfach über den Befehl *pip install xlrd* in einem, mit Python konfigurierten, Terminal. Beispielsweise über die Windows-Eingabeaufforderung, nachdem Python installiert wurde. Nach dem Installationsverfahren ist die Bibliothek auch schon einsatzbereit.

#### Anwendung

Für das Programm, welches im Rahmen der IPA entwickelt werden soll, gibt es drei wesentliche Aspekte, welche von der Library verwendet werden. Diese wären:

* Einlesen der Exceldatei
* Auswählen eines bestimmten Sheets
* Auslesen der Werte von bestimmten Zellen

Die oben genannten Funktionalitäten werden im Folgenden genauer erklärt. Eine Dokumentation über die gesamte Library ist unter <https://xlrd.readthedocs.io/en/latest/index.html> erreichbar.

Bevor die Funktionalitäten auch gebraucht werden können, ist es notwendig die Bibliothek am Anfang des Scripts zu importieren. Dies ist mit folgender Codezeile zu erreichen:

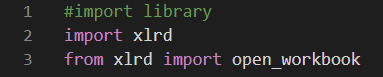


Abbildung : Import xlrd-Library

In Zeile zwei mit *import xlrd* wird zunächst die gesamte Library importiert. Mit der dritten Zeile wird speziell *open\_workbook* importiert. Dies ermöglicht, dass später im Code das Hauptobjekt *xlrd* weggelassen werden kann.

**Einlesen von Exceldateien**

Zum Öffnen von Exceldateien bietet die Library folgende Funktionalität:



Abbildung : Erzeugung eines Workbook-Objektes

Bei der Funktion *open\_workbook()* wird als Parameter der Filename (beziehungsweise der Filepfad, wenn die Datei nicht im selben Verzeichnis wie das Script ist) als String verlangt. Als Returnwert gibt die Funktion ein Workbook-Objekt zurück. Diese wiederspiegelt die Exceldatei. Im Beispiel oben wird das Objekt dann einer Variabel zugewiesen, welche nun die nötigen Funktionalitäten erhält, um ein bestimmtes Sheet zu wählen.

**Auswählen eines bestimmten Sheets**

Es gibt zwei verschiedene Vorgehensweisen, wie ein Worksheet gewählt werden kann. Einmal ist dies via Namen möglich oder via Index des Sheets.

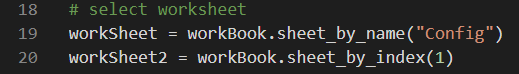


Abbildung : Erzeugung eines Worksheet-Objektes

Beide Methoden haben eine unterschiedliche Funktion, welche aufgerufen werden muss.

Wenn das Sheet anhand des Namens geladen werden soll, ist *sheet\_by\_name()* mit dem Namen als String als Parameter zu verwenden. Für den Aufruf via Index, nimmt man die Funktion *sheet\_by\_index()* mit der Index-Zahl des Sheets als Integer-Parameter.

Beide Funktionen geben ein Worksheet-Objekt zurück. Dies empfiehlt sich ebenfalls in einer Variablen abzuspeichern.

**Auslesen der Werte von bestimmten Zellen**

Für die Hauptfunktionalität der Library, also wie einzelne Zellenwerte ausgelesen werden können, ist es erforderlich ein Worksheet-Objekt zu haben. Ist dies Vorhanden erfolgt der Zellen-Zugriff wie folgt:

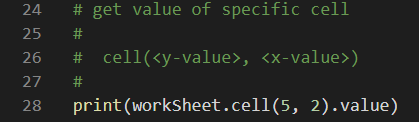


Abbildung : Zugriff auf Excel-Zellen

Die Funktion *cell()* erwartet zwei Integer-Parameter. Der erste Wert stellt die Y-Koordinate, oder auch den Reihen-Index, dar. Für die Wahl der Spalte der zweite Parameter zu verwenden. WICHTIG: Der Index für die Wahl der Spalte und der Reihe beginnen beide mit dem Wert 0!

Beispiele für Zellen-Aufrufe:

|  |  |
| --- | --- |
| Ansprechen einer Zelle | |
| Excel Schreibweise | Library Funktionsaufruf |
| A:1 | cell(0, 0) |
| A:4 | cell(5, 0) |
| C:1 | cell(0, 2) |
| D:8 | cell(9, 3) |
| F:12 | cell(13, 5) |

Tabelle : Zugriffarten auf eine Zelle

Damit der Wert der angesprochenen Zelle auch zurückgegeben wird, ist noch das entsprechende Property *value* anzugeben. Der Aufruf, *<worksheet>.cell(<y-value>, <x-value>).value*, ermöglicht es, den Wert einer bestimmten Zelle in einem Excelsheet auszulesen.

### Konventionen im Code

Für die Namengebung im Code wurde sich an die folgenden Konventionen gehalten. Allgemein gilt, dass anhand des Namens klar der Zweck der jeweiligen Variabel oder Funktion abgelesen werden kann.

#### Globale Variablen

Der Name einer globalen Variablen beginnt immer mit einem Unterstrich (\_).

Bsp.:

* \_workSheet
* \_metaDict

#### Konstanten

Bei Konstanten wird der gesamte Variablen-Namen in Grossbuchstaben geschrieben.

Bsp.:

* CONNECTIONS\_ENTRYPOINT\_X
* TESTSTEP\_PROPERTIES\_Y\_VALUE

#### Funktionen

Funktionen sind nach der Upper-Camel-Case-Notation benannt. Jedes Wort innerhalb des Namens beginnt mit einem Grossbuchstaben. Es sind ebenfalls keine Unterstriche enthalten.

Bsp.:

* PrintSummary()
* ConvertCoordinate()

#### Variablen

Für die Namensgebung der «normalen» Variablen wird die Lower-Camel-Case-Notation verwendet. Auch hier dürfen keine Unterstriche vorkommen. Allerdings wird der erste Buchstabe der Variable klein geschrieben. Ansonsten wird jedes neue Wort mit einem Grossbuchstaben begonnen.

Bsp.:

* tmpDevice
* firstProp

## Kontrollieren

### Testkonzept

#### Testumgebung

Die Tests werden auf derselben Umgebung ausgeführt, wie das Programm auch entwickelt wurde. Dafür stehet folgender Testumgebung zur Verfügung. Die Informationen stammen aus den Systeminformationen.

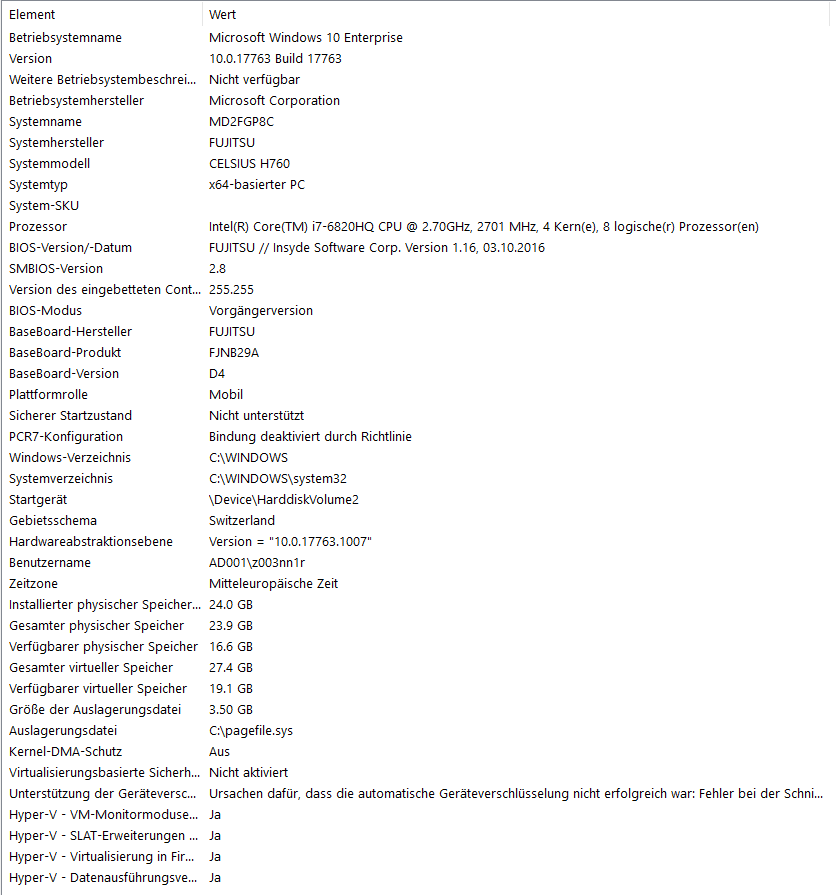


Abbildung : Systeminformationen

#### Testdaten

Die Input-Testdaten stammen aus der Datei «TsNetV1\_Example.xls». Dieses File beinhaltet einen klassischen Test, wie er auch später gebraucht werden könnte. Diese Datei kann für spezielle Tests auch abgeändert werden, so dass das Programm auf die Fehlerbehandlung getestet werden kann. Als Output-Daten werden die daraus generierten JSON-Files bewertet. Diese werden manuell vom Tester überprüft, ob diese den geforderten Resultaten aus dem Projektauftrag entsprechen.

#### Testablauf

##### Genereller Ablauf

Für das Testverfahren gibt es zwei verschiedene Arten von Tests. Einmal den Produkttest, welcher als Black-Box-Test gestaltet ist und einmal die Modultests, welche als White-Box-Tests aufgebaut sind.

Für beide Testarten werden die Testfälle vor der Implementation definiert. Das Testergebnis wird nach der Umsetzung des Programmes, beziehungsweise des Modules, überprüft. Treten dabei allenfalls Fehler auf, werden diese korrigiert und die entsprechenden Tests erneut durchgeführt.

##### Modultests

Der Aufbau des Programmes wurde in vier wesentliche Teile unterteilt:

* «meta»-Bereich
* «connections»-Bereich
* «test»-Bereich
* Fehlerbehandlung

Abgesehen von der Fehlerbehandlung wurden die anderen Teile in Module unterteilt. Der Grund, dass die Fehlbehandlung kein eigenes Modul ist, ist da es über das gesamte Programm verteilt eingesetzt wird und es somit nicht richtig abgrenzbar ist.

Es werden also für den «meta»-Bereich, «connections»-Bereich und für den «test»-Bereich immer vor der Implementation die Testfälle der Modultests definiert.

##### Produkttest

Mit dem Produkttest wird das gesamte Programm auf Fehler überprüft. Es wird geprüft, ob mit dem Inputfile auch das korrekte Outputfile, gemäss Projektauftrag erzeugt werden kann. Es wird ebenfalls getestet, ob bei Fehlern die Applikation korrekt reagiert und die geforderten Fehlermeldungen ausgibt.

### Modultests

#### «meta»-Daten-Tests

**Testfälle**

|  |  |
| --- | --- |
| **Test-ID: MT1.1** | |
| **Name** | Korrekte Struktur |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Die JSON-Struktur des meta-Bereiches stimmt mit den Anforderungen überein und ist in die zwei Abschnitte «testData» und «testObject» unterteilt |
| **Test-ID: MT1.2** | |
| **Name** | Korrekter Inhalt |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das JSON des meta-Bereiches wird korrekt erzeugt mit passender Struktur und den korrekten Daten aus dem Inputfile |
| **Test-ID: MT1.3** | |
| **Name** | Fehlender Inhalt |
| **Testvoraussetzung** | Im Excelfile fehlen Daten, welche im JSON-File gebraucht werden |
| **Testablauf** | 1. Im Inputfile werden einige Daten gelöscht, welche im JSON-File für den meta-Bereich benötigt werden 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine Warnung aus. Die fehlenden Daten werden im JSON leer gelassen. Alle vorhandenen Daten werden normal ausgeführt. |
| **Test-ID: MT1.4** | |
| **Name** | Fehlender meta-Bereich |
| **Testvoraussetzung** | Im Excelfile fehlt der meta-Bereich |
| **Testablauf** | 1. Im Inputfile wird der meta-Bereich entfernt 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine Fehlermeldung aus und wird beendet. |
| **Test-ID: MT1.5** | |
| **Name** | Datum-Formatierung |
| **Testvoraussetzung** | Korrektes Excelfile, in welchem ein Datum vorkommt |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm behält die Formatierung des Datums im Format(dd.mm.YYYY) auch in der JSON-Struktur bei. |

|  |  |
| --- | --- |
| **Test-ID: MT1.6** | |
| **Name** | Testcase spezifische Werte stimmen |
| **Testvoraussetzung** | Korrektes Excelfile |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Im Bereich der «testData» haben die Properties «testCaseName» und «testCaseDescription» jeweils die richtigen Werte und sind auf den jeweiligen Testcase angepasst |

Tabelle : Meta-Daten-Testfälle

**Testergebnis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | | **24.03.2020** | |
| **Testperson** | | **Timo Gloor** | |
| **Test-ID** | **Erwartung erfüllt?** | **Kommentar** | **Weitere Schritte** |
| **MT1.1** | Ja | Die Struktur des JSON-Formates stimmt. Korrekte Aufteilung in «testData» und «testObject» | Keine |
| **MT1.2** | Ja | Alle Daten werden korrekt eingelesen und entsprechend in der JSON-Struktur an den korrekten Orten ausgegeben | Keine |
| **MT1.3** | Ja | Es wird eine Warning-Meldung ausgegeben und der Inhalt im JSON leer gelassen | Keine |
| **MT1.4** | Ja | Eine Fehlermeldung wird ausgegeben und das Programm beendet | Keine |
| **MT1.5** | Ja | Das Datum wird im Format dd.mm.YYYY abgespeichert | Keine |
| **MT1.6** | Ja | Die spezifischen Werte für den jeweiligen Testcase stimmen | Keine |

Tabelle : Meta-Daten-Testergebnisse

#### «connection»-Daten-Tests

|  |  |
| --- | --- |
| **Test-ID: MT2.1** | |
| **Name** | Korrekte Struktur |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend   2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Die JSON-Struktur des connections-Bereiches stimmt mit den Anforderungen überein |
| **Test-ID: MT2.2** | |
| **Name** | Korrekter Inhalt |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile ist den Anforderungen entsprechend  2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das JSON des connections-Bereiches wird korrekt erzeugt mit passender Struktur und den korrekten Daten aus dem Inputfile |
| **Test-ID: MT2.3** | |
| **Name** | Fehlender Inhalt |
| **Testvoraussetzung** | Im Excelfile fehlen einige Daten, welche im JSON-File für den connections-Bereich gebraucht werden |
| **Testablauf** | 1. Im Inputfile werden einige Daten gelöscht, welche im JSON-File für den connections-Bereich benötigt werden 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine entsprechende Fehlermeldung aus und wird danach direkt beendet. |
| **Test-ID: MT2.4** | |
| **Name** | Variable Anzahl Devices |
| **Testvoraussetzung** | Korrektes Excelfile mit mehreren Devices |
| **Testablauf** | 1. Das Inputfile entspricht den Anforderungen 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm erkennt die variable Anzahl an Devices und liest alle angegebenen Devices korrekt ein. |

Tabelle : Connections-Daten-Testfälle

**Testergebnis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | | **25.03.2020** | |
| **Testperson** | | **Timo Gloor** | |
| **Test-ID** | **Erwartung erfüllt?** | **Kommentar** | **Weitere Schritte** |
| **MT2.1** | Ja | Die Struktur stimmt mit den Anforderungen überein. Auch die verschiedenen Klammern ([] und {}) stimmen | Keine |
| **MT2.2** | Ja | Der Inhalt der Daten wird korrekt ins JSON gebracht und auch die Datentypen stimmen überein | Keine |
| **MT2.3** | Ja | Fehlt der Wert eines Properties wird sofort eine Fehlermeldung ausgegeben und das Programm wird beendet | Keine |
| **MT2.4** | Ja | Das Programm liest die korrekte Anzahl Devices ein. Es erkennt, wann kein Device mehr übrig ist zum Lesen | Keine |

Tabelle : Connections-Daten-Testergebnisse

#### «test»-Daten-Tests

|  |  |
| --- | --- |
| **Test-ID: MT3.1** | |
| **Name** | Korrekte Struktur |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile wird angegeben und ist korrekt 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Die Struktur des erzeugten JSON der Tests stimmen mit den Anforderungen überein |
| **Test-ID: MT3.2** | |
| **Name** | Korrekter Inhalt |
| **Testvoraussetzung** | Ein korrektes Excelfile wird angeben und alle Daten stimmen |
| **Testablauf** | 1. Das Inputfile wird als Parameter angegeben und alle Inhalte stimmten 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Der erzeugte JSON-Inhalt für die Tests stimmt und die Daten entsprechen dem erwarteten Resultat. |
| **Test-ID: MT3.3** | |
| **Name** | Fehlender Inhalt |
| **Testvoraussetzung** | Im Excelfile fehlen Daten, welche im JSON-File für den Test-Bereich gebraucht werden |
| **Testablauf** | 1. Das Inputfile wird manipuliert, so dass bei den Testcases einige Daten fehlen 2. Das Programm wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine Fehlermeldung aus und wird danach direkt beendet |
| **Test-ID: MT3.4** | |
| **Name** | Fehlende Testcases |
| **Testvoraussetzung** | Im Excelfile sind keine Testcase-Sheets vorhanden, obwohl diese in der Overview-Tabelle aufgeführt sind |
| **Testablauf** | 1. Die Exceldatei wird so abgeändert, dass die Tests der Overview-Tabelle nicht als Sheets vorhanden sind 2. Die Applikation wird gestartet |
| **Erwartetes Resultat** | Das Programm gibt für jedes nicht existente Testcase-Sheet eine entsprechende Warnung aus. Es wird kein JSON-File erzeugt. In der Zusammenfassung wird entsprechend daraufhin gewiesen und die Applikation wird beendet. |

|  |  |
| --- | --- |
| **Test-ID: MT3.5** | |
| **Name** | Formatregeln |
| **Testvoraussetzung** | Das angegebene Testfile ist korrekt |
| **Testablauf** | 1. Das korrekte Excelfile wird als Parameter übergeben 2. Das Programm wird aufgerufen |
| **Erwartetes Resultat** | Die Daten im erzeugten JSON-File entsprechen allen Formatregeln, wie sie in der Aufgabenstellung definiert wurden |

Tabelle : Test-Daten-Testfälle

**Testergebnis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | | **26.03.2020** | |
| **Testperson** | | **Timo Gloor** | |
| **Test-ID** | **Erwartung erfüllt?** | **Kommentar** | **Weitere Schritte** |
| **MT3.1** | Ja | Die Struktur stimmt mit den Anforderungen überein. Für die Step-List wurden korrekterweise die []-Klammern verwendet | Keine |
| **MT3.2** | Ja | Der Inhalt der einzelnen Tests stimmt. Die Werte sind so übernommen, wie sie im Inputfile definiert sind | Keine |
| **MT3.3** | Nein | Kommentar:  Der Prozess wird nicht wie erwartet abgebrochen und das File dennoch erzeugt  Weitere Schritte:  Fehler überprüfen und beheben:  Fehler im Test-Daten-Modul Zeile 427:  if testCaseSheetName in \_workBook.sheet\_by\_index(i).name:  if testCaseSheetName == \_workBook.sheet\_by\_index(i).name: | |
| **MT3.4** | Nein | Kommentar:  Das Programm erstellt ein JSON-File, obwohl es nicht als Sheet existiert  Weitere Schritte:  Fehler überprüfen und beheben:  If-Abfrage vor dem Erstellen des JSON-Files hinzugefügt  if not errorInSheet: # Create sheet only if there is no error in it  #Create file | |
| **MT3.5** | Ja | Alle Formatregeln werden eingehalten und korrekt im JSON-File umgesetzt | Keine |

Tabelle : Test-Daten-Testergebnisse

**Nachtests**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | | **31.03.2020** | |
| **Testperson** | | **Timo Gloor** | |
| **Test-ID** | **Erwartung erfüllt?** | **Kommentar** | **Weitere Schritte** |
| **MT3.3** | Ja | Das Erstellen des JSON-Files wird abgebrochen und es wird mit dem nächsten Testcase weitergemacht. Eine entsprechende Meldung wird ausgegeben | Keine |
| **MT3.4** | Ja | Das Sheet wird nicht erstellt. Es wird eine Fehlermeldung ausgegeben und in der Zusammenfassung ebenfalls nicht angezeigt | Keine |

Tabelle : Test-Daten-Nachtests

### Produkttest

**Testfälle**

|  |  |
| --- | --- |
| **Test-ID: PT01** | |
| **Name** | Korrekte Generierung der Testcases |
| **Testvoraussetzung** | Ein korrektes Inputfile, wie auch ein bestehender Ordner für die Output-Files wird angegeben |
| **Testablauf** | 1. Das Import-Script wird via die Konsole aufgerufen 2. Als Parameter werden korrekte Dateipfade angegeben 3. Das Script wird ausgeführt |
| **Erwartetes Resultat** | Es treten keine Fehler auf. Vom Programm wird eine abschliessende Zusammenfassung über die erstellten JSON-Files ausgegeben. Die JSON-Files sind allesamt korrekt generiert und unter dem angegeben Verzeichnis mit korrektem Namen abgespeichert. |
| **Test-ID: PT02** | |
| **Name** | Inputfile existiert nicht |
| **Testvoraussetzung** | Der Pfad für die Eingabedatei ist falsch |
| **Testablauf** | 1. Das Import-Script wird via die Konsole aufgerufen 2. Als Parameter wird ein falscher Pfad für die Eingabedatei angegeben 3. Das Script wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine entsprechende Fehlermeldung aus und wird dann beendet. |
| **Test-ID: PT03** | |
| **Name** | Destination-Folder existiert nicht |
| **Testvoraussetzung** | Der angegebene Destination-Folder existiert nicht |
| **Testablauf** | 1. Das Import-Script wird via die Konsole aufgerufen 2. Als Parameter für die Ausgabedateien wird ein Verzeichnis angegeben, welches nicht existiert 3. Das Script wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine entsprechende Info aus und legt die Ordner an |

|  |  |
| --- | --- |
| **Test-ID: PT04** | |
| **Name** | File existiert bereits im Destination-Folder |
| **Testvoraussetzung** | Im angegeben Destination-Folder existiert bereits ein File mit dem gleichen Namen |
| **Testablauf** | 1. Das Import-Script wird via die Konsole aufgerufen 2. Als Parameter für die Output-Daten wird ein Verzeichnis angegeben, in dem es bereits ein File mit demselben Namen hat 3. Das Script wird ausgeführt |
| **Erwartetes Resultat** | Es wird vom Programm eine Warnung ausgegeben und das File wird überschrieben |
| **Test-ID: PT05** | |
| **Name** | Fehlende Testcase-Sheets |
| **Testvoraussetzung** | In der Overview-Tabelle sind Testcases angegeben, welche allerdings nicht als Sheet existieren |
| **Testablauf** | 1. Im Excelfile wird manuell ein Testcase-Eintrag ohne entsprechendes Sheet in der Overview-Tabelle hinzugefügt 2. Das Import-Script wird via die Konsole aufgerufen 3. Als Parameter werden korrekte Dateipfade angegeben 4. Das Script wird ausgeführt |
| **Erwartetes Resultat** | Das Programm gibt eine Information in der Konsole aus, dass der angegeben Testcase nicht gefunden wurde. Alle anderen Testcases werden dennoch erzeugt. Bei der Zusammenfassung wird eine entsprechende Information ausgegeben. |
| **Test-ID: PT06** | |
| **Name** | Fehlende Parameter |
| **Testvoraussetzung** | Beim Aufrufen des Programmes werden nicht alle Parameter mitgegeben |
| **Testablauf** | 1. Das Script wird ausgeführt und dabei nur einen Parameter mitgegeben |
| **Erwartetes Resultat** | Das Programm gibt direkt eine entsprechende Fehlermeldung aus und beendet sich dann anschliessend. |

Tabelle : Produkttestfälle

**Testergebnis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | | **27.03.2020** | |
| **Testperson** | | **Timo Gloor** | |
| **Test-ID** | **Erwartung erfüllt?** | **Kommentar** | **Weitere Schritte** |
| **PT01** | Ja | Alle Testcases werden korrekt generiert und sind im richtigen Verzeichnis zu finden | Keine |
| **PT02** | Ja | Es wird eine Fehlermeldung ausgegeben, dass das File nicht gefunden wurde und das Programm wird beendet | Keine |

|  |  |  |  |
| --- | --- | --- | --- |
| **PT03** | Ja | Es wird eine Warnung ausgegeben und der Ordner wird erstellt. Anschliessend sind die Dateien im neu erzeugten Verzeichnis zu finden. | Keine |
| **PT04** | Ja | Wenn bereits ein File mit dem gleichen Namen im Destination-Folder existiert, wird eine Warnung ausgegeben und das File wird überschrieben | Keine |
| **PT05** | Ja | Es werden nur die Files erzeugt, welche auch als Sheet vorhanden sind. Eine Warnung wird ausgegeben | Keine |
| **PT06** | Ja | Das Programm gibt einen Fehler aus und beendet sich anschliessend | Keine |

Tabelle : Produkttest Ergebnisse

## Auswerten

### Schlusswort

Als Projektmethode habe ich IPERKA gewählt. Da ich mit dieser Methode bereits in der Schule gearbeitet habe, hatte ich keine grossen Schwierigkeiten beim Planen des Projektes. Ich konnte ohne Probleme die einzelnen Tätigkeiten, die es zu erledigen galt, in die einzelnen Phasen des Projektes einteilen.

Bei der Planung für die Umsetzung verlief ebenfalls alles wie geplant. Die Variantenanalyse gab ein klares Resultat wieder und konnte dadurch die bevorzugte Methode wählen. Abgesehen von einem technischen Problem lief auch die Erstellung des Konzeptes gut.   
Nach einer jeweils kurzen Überlegungszeit waren auch schnell die einzelnen White- und Black-Box-Tests gefunden. Diese wurden in Modul- und Produkttests unterteilt.

Ich habe bereits vor der IPA mit dem TsNet und den dazugehörigen Excel-Testsheets gearbeitet. Daher kannte ich den groben Aufbau der Sheets schon ungefähr. Dies hat mir bei der Umsetzung des Programmes geholfen. Ich musste mich somit nicht lange in die einzelnen Sheets einarbeiten, sondern konnte schnell beginnen.

Ein weiterer Vorteil war, dass ich bereits mit der Programmiersprache Python vertraut war. Ich musste also nicht lange recherchieren, wie die Sprache überhaupt funktioniert.   
Auch auf JSON-Dateien bin ich im Verlauf meiner Lehre schon häufiger gestossen und wusste daher bereits, wie diese strukturiert sind und was es zu beachten gibt.

Lediglich die Dictionaries in Python waren für mich noch etwas Neues. Jedoch konnte ich nach einer kurzen Internet-Recherche auch damit gut umgehen und hatte keine weiteren Probleme.

Am meisten Zeit während der Implementierungs-Phase hat das Herausfinden der Algorithmen für den Daten-Zugriff gebraucht. Besonders der Algorithmus für das Test-Daten-Modul hat mich vor eine Herausforderung gestellt. Es hat etwas Zeit und viele falsche Resultate benötigt, bis ich auf eine Lösung kam, welche den korrekten Zugriff und Verarbeitung der Excel-Daten gewährleistet.

Dennoch bin ich froh, dass ich die Realisierungsphase plus minus so abschliessen konnte, wie ich es im Zeitplan vorgesehen hatte. Ich bin nie auf grosse Probleme gestossen, welche mir extrem viel Zeit geraubt haben und konnte den entsprechenden Meilenstein auch pünktlich erreichen.

Bei der Kontrolle mit den Produkt- und Modultests bin ich lediglich auf zwei Tests gestossen, welche nicht beim ersten Mal das erwartete Resultat lieferten. Die Fehler dazu konnten aber behoben werden und die Nachtests lieferten dann auch direkt ein positives Ergebnis.

Insgesamt kann ich sagen, dass meine IPA an sich ganz gut verlaufen ist. Ich bin nie auf ein Problem gestossen, welches ich nicht selbst lösen konnte oder mich viel Zeit gekostet hatte. Natürlich sind vereinzelt Schwierigkeiten aufgetreten, aber diese konnte ich dennoch alle lösen. Ich war stets sehr gut im Zeitplan. Eigentlich auch immer ein wenig voraus, somit bin ich auch nie in eine stressige Phase gekommen.   
Als Endprodukt habe ich eine Applikation, welches die Aufgabe erfüllt, die in der Aufgabenstellung gefordert wird. Das Ganze wurde in der geforderten Zeit erreicht und kann somit zufrieden sein, dies so erreicht zu haben.

Für ein nächstes Mal würde ich eventuell etwas mehr Zeit für die Implementierung planen. Somit könnte ich den Code nicht nur zweckgemäss schreiben, sondern hätte auch mehr Zeit für eine saubere Planung und Gliederung. Ansonsten würde ich gleich vorgehen, wie ich es jetzt getan habe.

## Glossar

|  |  |
| --- | --- |
| **Begriff** | **Erklärung** |
| Aktivitätsdiagramm | Stellt die Reihenfolge und Abläufe einzelner Aktionen graphisch dar |
| Applikation | Hier: Software, die in einem Controller die eigentlichen Steuer- und Regelaufgaben übernimmt |
| BACnet | Building Automation Control Network: standardisierte Definition von Objekten, Services und Kommunikationsregeln für Gebäudeautomation |
| Black-Box-Test | Die Testfälle werden anhand der Spezifikation beziehungsweise der Anforderungen abgeleitet. Dabei hat der Ersteller der Tests keine Kenntnis über das Innere der Struktur |
| code.siemens.com | Auf dem Siemens-Intranet verfügbare Instanz von GIT, hier verwendet für Dokumentation und Source Code des TsOpen-Projekts |
| Controller | Für Steuerung und Regelung von Prozessen in der Gebäudeautomation eingesetztes Gerät, das einen Mikroprozessor, diverse Ein- und Ausgänge sowie Kommunikationsschnittstellen besitzt. Ein Controller besitzt Objekte, über die Daten mit dem Benutzer oder mit dem Prozess ausgetauscht werden können |
| Dictionary | Objekt zur Sammlung von Daten. Arbeitet mit dem Key-and-Value-Prinzip |
| EDE | Engineering data export, gemäss BACnet standardisiertes csv-File, das die einzelnen Objekte eines Controllers auflistet mit u.a. Namen, Beschreibung und einer ID, mit der das Objekt im Controller angesprochen werden kann |
| GIT | Freie (Open Source) Versionsverwaltung von Dateien |
| IPERKA | Projektmanagementmethode, welches das Projekt in sechs unterschiedliche Projektphasen unterteilt:   * Informieren * Planen * Entscheiden * Realisieren * Kontrollieren * Auswerten |
| JSON | Kurz für JavaScript Object Notation. Lesbares Datenformat, zum Abspeichern von Objekten und Daten |
| Modul | Teil bzw. Element des gesamten Systems |
| Modultest | Test zum Testen eines Modules und deren Funktionalitäten. Wird hier als White-Box-Test aufgebaut |
| Objekte | Hier: Fest definierte Daten und Funktionen, mit denen die Applikation eines Controllers auf den Prozess und auf Benutzereingriffe zugreifen kann. Objekte dienen zur Anzeige von Prozesszuständen, zur Eingabe von Sollwerten durch Benutzer, zum Ansteuern von Ausgängen, mit denen der Prozess beeinflusst wird. Sie werden auch zur Simulation und zum Testen verwendet |
| Produkttest | Testfälle zum Testen der gesamten Applikation. Wird hier als Black-Box-Test aufgebaut |
| Range | Hier: Bereich innerhalb eines Excel-Sheets |
| SSD | Solid-State-Drive. Externer Datenspeicher, auf dem für das Projekt täglich Backup-Dateien gespeichert werden. Dient der Datensicherung |
| Struktogramm | Darstellungsmethode zur Veranschaulichung von Abläufen innerhalb eines Programmes oder Prozedur |
| Testfall | Auflistung von Testschritten, hier einzelne Aspekte einer Applikation, z.B. Aussentemperaturabhängigkeit |
| Testschritt | Auflistung von einzelnen Anweisungen «Tasks», die zum Test durchgeführt werden müssen, und der dazu gehörenden erwarteten Ergebnisse «Expected»,  z.B. «Task»: mache Drahtbruch am Sensor  «Expected»: Anzeige der Fehlermeldung 0005 am Display |
| Testscript | Testfall, der automatisiert abläuft |
| Test-Spezifikation | Auflistung von Testfällen, hier zum Test einer Applikation, z.B. Heizkreisregelung |
| TsNet | Bestehendes, aktuell genutztes Test-Framework zum automatisierten Testen von Applikationen der Gebäudeautomation |
| TsOpen | Neues, derzeit entwickeltes Test-Framework zum automatisierten Testen von Applikationen der Gebäudeautomation |
| White-Box-Test | Ermittlung der Testfälle mit Kenntnis über den Code und deren Struktur |

Tabelle : Glossar

**Anmerkung:** Einige Begriffe wurden aus der Aufgabenstellung entnommen und hier mit allen anderen Begriffen zusammengefasst. Dadurch kann ein gemeinsames Glossar erstellt werden.

## Quellen

draw.io, Abgerufen am 19.03.2020 von draw.io: <https://www.draw.io>

Hans-Ulrich, S., *hus Strutkogrammer*, Abgerufen am 20.03.2020 von struktogrammer.ch: <http://www.struktogrammer.ch/>

Machin, S. (2018), *xlrd documentation*, Abgerufen am 18.03.2020 von readthedocs.io: <https://xlrd.readthedocs.io/en/latest/>

PkOrg (2020), Zuletzt abgerufen am 02.04.2020 von pkorg.ch: <https://www.pkorg.ch/>

Python Software Foundation (10.03.2020), *Download Python,* Abgerufen am 31.03.2020 von python.org: <https://www.python.org/downloads/>

Python Software Foundation (16.06.2019), *os.path – Common pathname manipulations,* Abgerufen am 24.03.2020 von python.org: <https://docs.python.org/3.4/library/os.path.html>

Stackoverflow (2012), *Reading date as a string not float from excel using python xlrd*, Abgerufen am 24.03.2020 von stackoverflow.com: <https://stackoverflow.com/questions/13962837/reading-date-as-a-string-not-float-from-excel-using-python-xlrd>

tgabathuler, *Iperka,* Abgerufen am 18.03.2020 von tgabathuler.ch: <https://www.tgabathuler.ch/Iperka/Einfuehrung.html>

w3schools, *Python Dictionaries,*Abgerufen am 24.03.2020 von w3schools.com: <https://www.w3schools.com/python/python_dictionaries.asp>

w3schools, *Python JSON*, Abgerufen am 24.03.2020 von w3schools.com: <https://www.w3schools.com/python/python_json.asp>

w3schools, *Python Tutorial,*Abgerufen am 24.03.2020 von w3schools.com: <https://www.w3schools.com/python/>

## Anhang

Im PkOrg ist der Sourcecode zusammen mit dem Beispiel-Input-File «TsNetV1\_Example.xls» als Anhang zu finden.