PSI

Paul Scherrer Institut

Inventurscript zum Abgleich von installierter Hardware mit einer zentralen Datenbank

Individuelle Praktische Arbeit 2025

Wernle Yannick
24. März 2025

Versionierung

Version	Datum	Autor	Bemerkung
1.0	04.03.2025	Yannick Wernle	Dokumentstruktur, Teil 1, Informieren, Beginn Planen & Arbeitsjournal
1.1	05.03.2025	Yannick Wernle	Planen weitgehend abgeschlossen, Entscheidungsmatrix erstellt & Arbeitsjournal
1.2	07.03.2025	Yannick Wernle	Beginn Realisation, Grundstruktur & Konfigurationsdateien, Arbeitsjournal
1.3	10.03.2025	Yannick Wernle	Realisierung Start per Parameter, Logfunktion, Pingtest, Arbeitsjournal
1.4	11.03.2025	Yannick Wernle	Abfrage Inventory, Arbeitsjournal
1.5	12.03.2025	Yannick Wernle	Abfrage ILO, Scriptmessages, Switch MAC/Serial, Arbeitsjournal
1.6	14.03.2025	Yannick Wernle	JSON & CSV begonnen zu dokumentieren, Arbeitsjournal
1.7	17.03.2025	Yannick Wernle	Dokumentierung Dateispeicherung, Arbeitsjournal, Hilfestellungen
1.8	18.03.2025	Yannick Wernle	Fehlerbehandlung, Passwortsicherung, Testfälle überarbeitet, Arbeitsjournal
1.9	19.03.2025	Yannick Wernle	Arbeitsjournal, Abschluss Kontrollieren
1.10	21.03.2025	Yannick Wernle	Abschluss Auswertungsphase, Dokumentation nachführen --> ILO & Inventory in Informierungsphase nachgetragen, Kurzfassung, Struktur und Fehler angepasst
2.0	24.03.2025	Yannick Wernle	Rechtschreibung, Grammatik, Abgabe

Tabelle 1: Versionierung

Verteiler

Person	Funktion	Kontaktdaten
Jan Hohenheim	Hauptexperte	jan@hohenheim.ch
Gajanthan Vasantharuban	Nebenexperte	gajanthan.vasantharuban@gmail.com
Helge Brands	Fachvorgesetzter	helge.brands@psi.ch
Alfred Albisser	Berufsbildner	alfred.albisser@psi.ch
Yannick Wernle	Kandidat	yannick.wernle@psi.ch

Tabelle 2: Verteiler

Dokumentinformationen

Dokumentname	IPABericht_YannickWernle_ Inventurscript
Erstelldatum	04.03.2025
Abgabedatum	24.03.2025 / 13:00 Uhr
Startblock	2 (03.03 – 07.03.2025)

Tabelle 3: Dokumentinformationen

Vorwort

Die Individuelle Praktische Arbeit (IPA) ist die letzte, aber auch eine der wichtigsten Prüfung in der Lehre als Informatiker. Innerhalb von 10 Tagen zeigt der Kandidat seine Kompetenzen und baut eine Applikation komplett von A-Z anhand einer individuellen Aufgabe auf, mitsamt Planung, Implementation und Dokumentation. Im besten Fall ist sie auch dem Lehrbetrieb nach der Durchführung noch von nutzen.

Die IPA darf ich bei Helge Brands durchführen, bei dem ich seit letztem August sehr viel dazugelernt habe – nicht nur im Bereich Programmieren, sondern auch viel grundsätzliches Wissen über die Informatik. Mit seiner Hilfe konnte ich im Rahmen einer Test-IPA mich gut auf die IPA vorbereiten und wusste, was auf mich zukommt.

Mit diesem IPA-Bericht will ich nicht nur zeigen, dass ich die Anforderungen an das EFZ erfülle, sondern auch dass ich verstanden habe, womit ich überhaupt arbeite.

Die Arbeit entsteht zwischen dem 04.03.2025 und dem 24.03.2025, unterbrochen von jeweils 1.5 Schultagen pro Woche.

Danksagung

Diese IPA ist das Ende meiner der Lehre als Informatiker am Paul Scherrer Institut. An der Ausbildung ist aber nicht nur der Lernende beteiligt, sondern auch viele engagierte und kompetente Mitarbeiter, Fachvorgesetzte und Berufsbildner. Für dieses Engagement, das meistens neben der normalen Arbeit erbracht wurde, möchte ich mich herzlich bei allen Beteiligten bedanken.

Besonders möchte ich meinem Fachvorgesetzten Helge Brands danken, der mich optimal auf diese IPA vorbereitet hat. Weiterhin möchte ich mich auch bei meinen ehemaligen Praxisbildnern Thomas Attinger, Hebatallah Mostafa, Markus Winter und meinem Berufsbildner Alfred Albisser bedanken. Trotz des etwas chaotischen Berufsbildnerwechsels mitten in der Lehre haben sie sich für die bestmögliche Ausbildung und Beteiligung an Aufgaben und Wissen eingesetzt. Vielen Dank für diese lehrreiche, spannende Lehre!

Inhalt

1	Teil 1 – Umfeld und Ablauf		7
	1.1	Aufgabenstellung	7
	1.1.1	Titel	7
	1.1.2	Ausgangssituation	7
	1.1.3	Detaillierte Aufgabenstellung	7
	1.1.4	Mittel und Methoden	10
	1.1.5	Vorkenntnisse	10
	1.1.6	Vorarbeiten	10
	1.1.7	Verwendete Firmenstandards	10
	1.1.8	Neue Lerninhalte	10
	1.1.9	Arbeiten in den letzten 6 Monaten	10
	1.2	Projektorganisation	11
	1.2.1	Organigramm	11
	1.2.2	Arbeitsumfeld	11
	1.2.3	Datensicherung und Versionsverwaltung	12
	1.2.4	Projektmanagementmethode	13
	1.3	Zeitplanung	15
	1.3.1	Arbeitstage	15
	1.4	Meilensteine	16
	1.5	Zeitplan	17
	1.6	Arbeitsjournal	18
2	Teil 2 – Projektdokumentation		33
	2.1	Kurzfassung des IPA Berichts	33
	2.1.1	Ausgangslage	33
	2.1.2	Vorgehen	33
	2.1.3	Ergebnis	33
	2.2	Informieren	34
	2.2.1	Projektumfeld	34
	2.2.2	Systemgrenzen & Schnittstellen	34
	2.2.3	Anforderungsanalyse	35
	2.2.4	Programablaufplan	36

2.2.5	Integrated Lights Out (ILO)	38
2.2.6	Inventory.psi.ch	39
2.3	Planen	40
2.3.1	Testkonzept	40
2.3.2	Testfälle	41
2.3.3	Verwendete Technologien	48
2.3.4	Konfigurationsdatei(en)	48
2.3.5	Parameter	50
2.3.6	Programmablaufplan detailliert	52
2.4	Entscheiden	53
2.5	Realisieren	54
2.5.1	Grundstruktur	54
2.5.2	Modulversion	55
2.5.3	Generierung der Konfigurationsdatei	56
2.5.4	Start via Parameter	57
2.5.5	Logfunktion	58
2.5.6	Pingtest	59
2.5.7	Abfrage von Inventory	59
2.5.8	Abfrage von ILO	61
2.5.9	Dateispeicherung	63
2.5.10	Hilfestellungen	65
2.5.11	Fehlerbehandlung	66
2.5.12	Sicherung des Passwortes	69
2.6	Kontrollieren	70
2.6.1	Testprotokoll 1 - 19.03.2025	70
2.7	Auswerten	72
2.7.1	Technisches Fazit	72
2.7.2	Zukunftsaussichten	72
2.7.3	Selbstreflexion & Persönliches Fazit	73
3	Literaturverzeichnis	74
4	Abbildungsverzeichnis	75
5	Tabellenverzeichnis	76
6	Codesnippetverzeichnis	77

7	Glossar	78
8	Abkürzungsverzeichnis	79
9	Selbstständigkeitserklärung	79
10	Anhang	80
10.1	Protokoll Erster Expertenbesuch	80
10.2	Protokoll Zweiter Expertenbesuch	81

Darstellung

Im Dokument werden für die Einheitlichkeit die folgenden Standards verwendet:

Lorem Ipsum	Die Standardschrift des PSI (Calibri) ohne spezielle Formatierung wird für selbstverfasste Texte verwendet.
«Lorem Ipsum»	Zitate werden mit «» gekennzeichnet.
Lorem Ipsum	**Fett markierter Text hat besondere Wichtigkeit.**
Lorem Ipsum	Dick unterstrichene Wörter kennzeichnen einen Eintrag im Glossar
Lorem Ipsum	Unterstrichene Wörter kennzeichnen Links.
Lorem Ipsum	Weisser Text auf schwarzem Grund kennzeichnet Code, resp. Codeausschnitte.

Abbildung 1: Beispiel Abbildung

```
QClipboard *clipboard = QApplication::clipboard();
```

Codesnippet 1: Beispiel Code

Definition	Wert
A	b

Tabelle 4: Beispieltabelle

1 Teil 1 – Umfeld und Ablauf

1.1 Aufgabenstellung

1.1.1 Titel

Inventurscript zum Abgleich von installierter Hardware mit einer zentralen Datenbank (inventory.psi.ch)

1.1.2 Ausgangssituation

Die in unseren Anlagen (SLS und SwissFEL) vorhandenen Serversysteme besitzen ein Wartungsinterface. Dieses Wartungsinterface (iLO von HPE) ist ein eigener Rechner im Rechner und stellt, externen Systemen, die Möglichkeit zur Verfügung auf Hardwareebene über Netzwerk Wartungsaufgaben (Installation, Fehlerdiagnosen und Inventur) durchzuführen. Um einen Überblick über die bestehenden Anlagen zu haben und Änderungen an den Serversysteme schnell erfassen zu können, benötigen wir ein Tool welches aus allen Servern die wichtigen Adressierungsnummern (MAC Adressen) und die Seriennummern der Hauptkomponenten (Mainboard) ausliest und ein Reporting erstellt. Dieses Script muss in Powershell geschrieben werden, weil hier herstellerseitig eine Bibliothek zur Auslese der benötigten Parameter schon vorhanden ist.

Zur Ermittlung der gültigen Zielsysteme soll anhand von Suchstrings die für die Anlage gültigen Systeme aus dem Inventory (Web basierter Datenbankdienst) ermittelt und abgefragt werden. Die abgefragten Serversysteme und ermittelten Daten sollen dann mit den erfassten Daten im Inventory basismässig verglichen werden.

1.1.3 Detaillierte Aufgabenstellung

Die Hauptaufgabe ist das Erstellen eines Powershellscripts, dass mit Informationen aus der Inventorydatenbank (Webservice) oder der Implementierung einer Offlineversion Inventurdaten erfasst.

Die Verbindung zum Inventory soll dazu verwendet werden die, nach der Namenskonvention eingetragenen Serversysteme zu finden und ein Abgleich der vorhandenen Seriennummern und MAC Adressen zu machen. Der Report enthält dann die Unterschiede zum Inventory . Um das System für den Anwender transparent zu machen sollen die Zwischenschritte in Dateien zwischengespeichert werden, sodass man auch ohne den Webzugriff auf das Inventory arbeiten könnte. In einer frei wählbaren Konfigurationsdatei werden die Suchstrings spezifiziert nach denen gesucht werden kann.

Die Ergebnisse werden tabellarisch in einer Datei gespeichert, um sie kontrollieren zu können. Der automatsche Update ist nicht wünschenswert, da hier zu viele Details sonst implementiert werden müssten.

Anforderungen:

a. Hilfe

- Anzeigen einer Hilfeseite mit Parametern bei
  - i. Anforderung(--help ; -h ; /?; PowerShell help command [Benutzerführungsstil Windows/Unix])
  - ii. Bei fehlerhaften Parametern wird die Hilfeseite angezeigt und die Fehler zusätzlich beschrieben
  - iii. Bei Fehlern in den Konfigurationsdateien wird der Pfad und das Problem signalisiert und die Hilfe angezeigt.
  - iv. Die Hilfeseite wird bei den Anforderungen angezeigt, als Hilfe für den Benutzer.

b. Initialisierung

- Die Bibliotheksversionen (während der Entwicklung und bei der Ausführung) sollen verglichen werden. Nur wenn die Versionen unterschiedlich sind, taucht ein Hinweis dazu auf.
- Es sollen Konfigurationsdateien in JSON initialisiert werden können. Es gibt 2 Arten davon:
  - i. Eine vorgegebene Datei mit einer Standardkonfiguration.
  - ii. Eine leere Konfigurationsdatei, wo die Parameter vom Benutzer selbst eingefüllt werden müssen.
- Parameter ohne eine Konfigurationsdatei sollen nur über die Kommandozeile weitergegeben werden können. Welche Parameter das sind muss der Kandidat entscheiden.

c. Weitere Konfigurationsparameter des Powershellscripts

- Logparameter
  - i. Ein und Ausschaltung der Logfunktion
  - ii. Level des Detailgrades (z.B: Alarmwerte 1-10, Warnwerte 11-20, Infowerte 21-30)
  - iii. Zieldatei in welche die Loginformationen geschrieben werden müssen
- Verbindungsparameter
  - i. Suchstrings zur Datenbanksuche mit Feldangabe zur Remote-Management Host Abfrage
  - ii. Direkte Hostnamen des / der Zielsysteme (Standalone-Lösung zur Vorinventarisierung)
  - iii. Dateipfad zur Konfiguration
- Dateiparameter
  - i. Ziel der Report-Datei
  - ii. Suchpfade für Dateien
- Abfrageparameter
  - i. Pingtest: soll ausschaltbar sein
  - ii. Seriennummern: das Speichern (CSV-Dateien) von Seriennummern muss ausschaltbar sein
  - iii. Macadressen: das Speichern (CSV-Dateien) von Macadressen muss ausschaltbar sein

d. Ausführen einer Verbindungskontrolle und Problemlösung

- Einfacher Ping zur Sicherstellung der Erreichbarkeit
- Erstellung von Script-Messages mit Zeitstempeln
- Erstellung eines Logfiles zur Diagnose (TXT Format) des Laufzeitverhaltens

e. Auslese der Parameter
- Verbindungsaufbau mit Nutzerinformationen auf dem Bildschirm
- Weiteres schreiben des Logfiles zur Diagnose (TXT Format/log)
- Erstellung eines Reportfiles (JSON Format)
- Nutzinformationen (Hostname + ausgelesene Seriennummer + Anzahl an Netzwerkinterfaces) in einem Excel importfähigen Format (CSV Format)

f. Tests : Der Kandidat muss für das Testen des Tools Laufzeittests erstellen. Folgende Serversysteme sollen dabei getestet werden:
- Laufzeittest mit den Serversystemen DL380 Gen8/Gen9/Gen10/Gen11
- Laufzeittest mit DL20 Gen10 als nicht Standartsystem

g. Prozessoptimierung
- Anpassung der zur Inventarisierung notwendigen Schlüsselbegriffe, diese müssen vom Kandidaten bestimmt werden
- Optimierung der Datenstruktur in den Exportdateien zum Abgleich der Datenbankinformationen (Inventory). Diese ist vom Kandidaten durchzuführen.

Anwendungsbeispiele:
a. lokale Anwendung ohne Datenbankanbindung (Hardwareverantwortliche Person oder Administrator)
Das zu erstellende Script wird mit den werkseitig (HPE) vordefinierten Parametern (Hostname, User und einem Kennwort z.B. mit Barcode/QR Code Scanner eingelesen[nicht Teil der IPA]) konfiguriert um Inventardaten aus einem einzelnen Rechner automatisiert auszulesen

b. wöchentlicher automatische Ausführung (Administrator)
Das zu erstellende Script liest seine Konfiguration ein und sucht in der Inventurdatenbank nach den installierten Servern (Namenskonvention). Das Resultat dieser Suche (Servern in einem Beschleuniger) enthält unter anderem die Hostnamen der Managementinterfaces welches zur Inventur abgefragt werden kann. Die gespeicherten Daten ermöglichen eine Rückverfolgung der eingesetzten Hardware in den abgefragten Servern.

Personengruppe die dieses Script anwenden:
Das zu erstellende Script wird nur von den Administratoren der Kontrollgruppe und von dem Verantwortlichen für die Beschleunigerhardware gebraucht und benutzt. Es wird verwendet um die manuelle Erfassung zu automatisieren und um Änderungen, die in unseren Anlagen durch Piketteinsätze durchgeführt werden besser nachvollziehen zu können.

weitere Datenverarbeitung [nicht Teil der IPA]
a) das Powershellscript hat einen neuen Rechner ausgelesen:
Die erfassten Daten können per Importfunktion in die Datenbank übernommen werden.
b) es wurde eine Änderung am Wochenende im Pikettdienst vorgenommen:
Man sucht anhand des Datums die benötigen Logfiles und überprüft die Hardwareänderung und ob die Informationen mit dem Lager übereinstimmen.
c) Änderungsnachverfolgung von grösseren Umbauarbeiten:
Hier kann durch Abgleich von Serienummern und MAC Adressen die exakte Position im Server und auch im Netzwerk nachvollzogen werden.

1.1.4 Mittel und Methoden

Software:
- Powershell
- HPEiLOCmdlets

Hardware:
- 4 Server HPE DL380 Gen8/9/10/11
- 1 Server HPE DL20 Gen10

Services:
- inventory - Datenbank /Webzugriff
- git Repository (intern oder extern)

1.1.5 Vorkenntnisse

- Powershell
- Netzwerkdiagnose (Wireshark)
- iLO
- http/https Protokoll
- Dateiformate

1.1.6 Vorarbeiten

In der Test-IPA wurden die verschiedenen Versionen von HPEiLOCmdlets für die vorhandene Servermodelle DL380 Gen8/9/10/11 getestet. Diese Vorarbeit war notwendig, weil die älteren Versionen zum Teil sehr unterschiedlich auf die verschiedenen HPE Generationen reagiert haben.

Ausserdem hat der Lernende selbstständig die Prüfungsdokumente studiert und basierend auf der Wegleitung seine Vorlagen für die Dokumentation und den Zeitplan erarbeitet.

1.1.7 Verwendete Firmenstandards

Am PSI wurde innerhalb des letzten Jahres ein neues Logo und somit auch neue Firmenstandards erstellt. Diese Firmenstandards werden in der IPA verwendet: Templates & Styleguide | PSI
Ausserdem wird die Abfrage der Server anhand des Hostnamen gemacht, welche einer Namenskonvention der Abteilung entsprechen: SwissFEL-Controls-NamingConvention-01.pdf

1.1.8 Neue Lerninhalte

- Suchanfrage an ein externes System
- Verwendung einer Webschnittstelle
- Abfangen zusätzlicher Fehlerquellen
- Neben JSON auch CSV als Datenformat, um in Excel zu importieren

1.1.9 Arbeiten in den letzten 6 Monaten

- Weiterentwicklung (Markierfunktion) eines Widgets in unserem Displaymanager (Qt/C++)
- Entwicklung von Userpanels aus Expertenpanels zur Steuerung von Servomotoren (Benutzung des Displaymanagers)
- Test-IPA als Vorbereitung

1.2 Projektorganisation

1.2.1 Organigramm

Es sind neben dem Kandidaten noch 4 weitere Personen direkt am Projekt beteiligt:

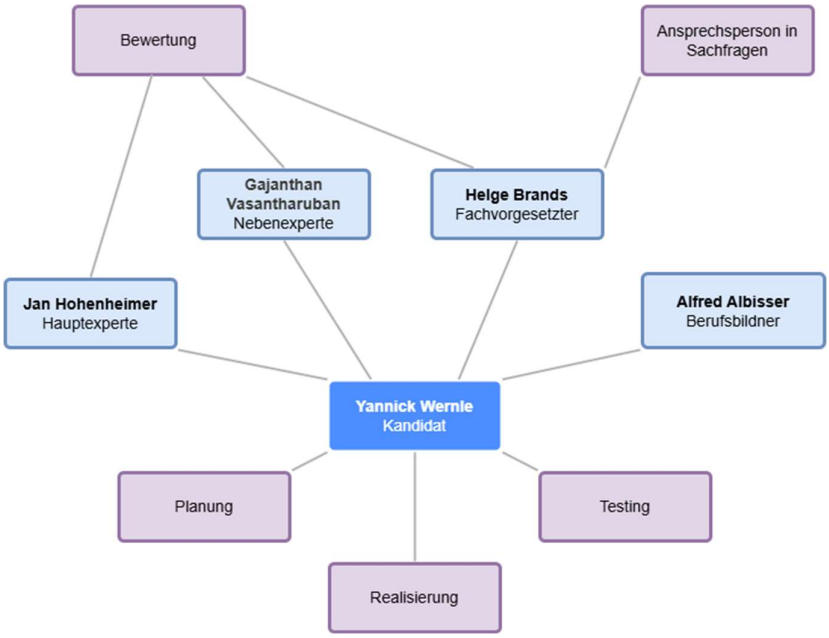


Abbildung 2: Organigramm

1.2.2 Arbeitsumfeld

Der Kandidat arbeitet jeweils am Montag, Dienstag, Mittwochnachmittag und am Freitag. Am Mittwochmorgen und am Donnerstag ist er aufgrund von Berufsschule und Berufsmatura abwesend.

Er arbeitet während der ganzen IPA am PSI, genauer gesagt im Gebäude WBGB/009, wo er auch sonst arbeitet. Um etwaige Personen darauf hinzuweisen, dass er nicht verfügbar ist, hat er im Outlook einen Blocker für die gesamte IPA-Zeit eingetragen und ist im Microsoft Teams als «Nicht Stören» gekennzeichnet.

Verwendete Software & Tools

- Microsoft Excel (Zeitplan & Entscheidungsmatrix)
- Microsoft Word (Dokumentation/IPA-Bericht)
- Microsoft Visual Studio Code (Entwicklungsumgebung)
  - PowerShell Erweiterung von Microsoft
- Git Extentions
- Draw.io
- Github
- Regexr.com
- Inventory.psi.ch

Verwendete Server

Der Kandidat hat aufgrund seiner Aufgabenstellung den Auftrag, das Script gegen die folgenden Server zu testen:

- gfa-sioc-cs-dev (ProLiant DL380 Gen8)
- sf-sioc-cs-64_defect (ProLiant DL380 Gen9)
- gfa-sioc-cs-de3 (ProLiant DL380 Gen10)
- gfa-sioc-cs-de4 (ProLiant DL380 Gen11)
- dl20test (ProLiant DL20 Gen10)

1.2.3 Datensicherung und Versionsverwaltung

Alle Daten und Dateien des Projekts werden auf Github abgelegt. Das Repository ist hier erreichbar: yativilli/hpeilo_inventoryscript. Es wurde nicht das PSI-interne Git verwendet, da dies bald von Gitlab auf Gitea migriert werden soll. Ausserdem beinhaltet das Script keinerlei vertrauliche Daten, weswegen es unbedenklich ist, dies ausserhalb dieses gesicherten Rahmens zu verwenden.

Die Software wird mindestens einmal täglich durch Commits auf das Repository gesichert – Git lässt anhand dieser Commits eine genaue Versionsverwaltung zu, weswegen diese auch verwendet wird.

Abbildung 3: Ausschnitt Github Commits

Der IPA-Bericht wird einerseits lokal auf dem Netzlaufwerk des Lernenden gespeichert und aktiv bearbeitet, andererseits wird er auf dem OneDrive der Schule ebenfalls einmal täglich abgelegt.
Diese doppelte Speicherung garantiert, dass weitgehend unabhängig der Systeme des PSI arbeiten kann – solange Github oder mein Netzlaufwerk erreichbar sind, kann alles andere ausfallen oder ausgetauscht werden.

Als weiteres Element werden auch die Issues von Github verwendet – diese lassen sich optimal nutzen, die einzelnen Implementierungsschritte zu planen und den Zeitplan im Auge zu behalten. Es wurde aber darauf verzichtet, für jedes Issue einen eigenen Branch zu erstellen, da dies unnötig viel Arbeit und Zeit fressen würde – da dies keine Gruppenarbeit ist, muss man auch keine Merge-Konflikte oder andere Konflikte befürchten.

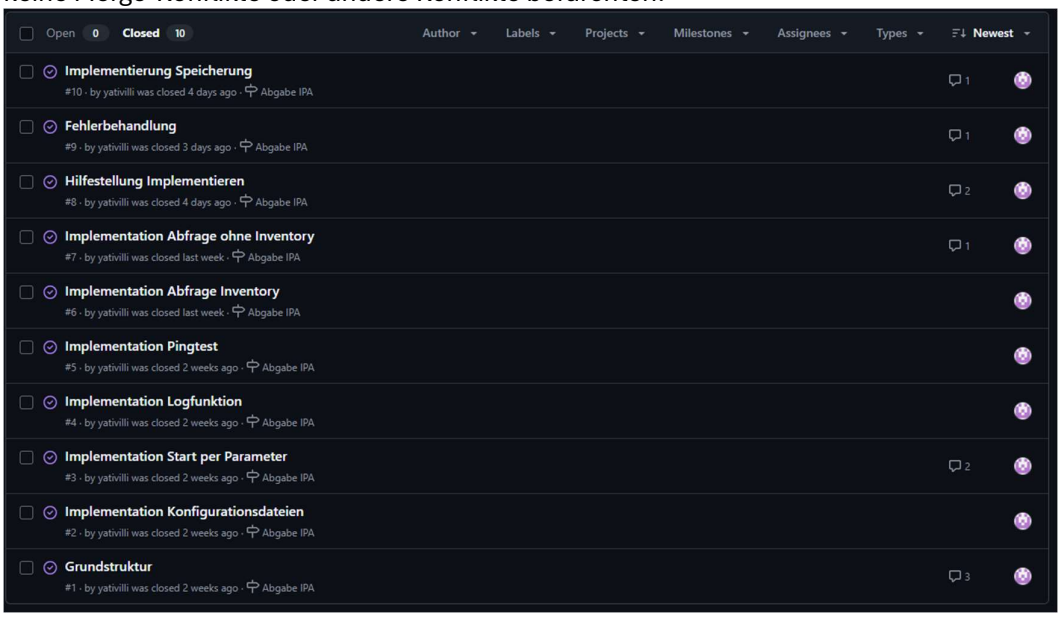


Abbildung 4: Git Issues

1.2.4 Projektmanagementmethode

Um die praktische Arbeit reibungslos durchführen zu können, hat sich der Kandidat dazu entschieden, die Projektmanagementmethode IPERKA zu verwenden.

IPERKA

IPERKA ist ein Akronym, welches für «Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Auswerten» steht. Die Projektmanagementmethode ist nach dem Wasserfallprinzip aufgebaut, was bedeutet, dass die Phasen aufeinander folgen und nicht iterativ sind.

Die Phase **Informieren** ist dafür gedacht, sich alle nötigen Informationen über das Projekt im Vorfeld bereits zu erfassen – bspw. welche Schnittstellen es gibt, wo die Grenzen liegen und welche Anforderungen es gibt.

Die Phase **Planen** ist wie am Namen erkennbar fürs Planen da – dazu gehören unter anderem Zeitplan oder Programmablaufplan.

Die Phase **Entscheiden** ist dafür gedacht, allfällige wichtige Entscheidungen, die getroffen werden müssen, bevor das Projekt realisiert wird, strukturiert und objektiv gefällt werden.

Die Phase **Realisieren** ist die Umsetzung des Projekts in die Wirklichkeit.

Darauf folgt die Phase **Kontrollieren** – dort wird überprüft, ob das Produkt allen Anforderungen entspricht und getestet, ob es auch funktionstüchtig ist.

Als abschliessende Phase ist die **Auswertung** gedacht – dort wird reflektiert, was am Projekt gut und schlecht gelaufen ist, wo es sowohl für die Beteiligten als auch das Produkt noch Verbesserungspotential gibt. Am Ende dieser Phase ist das Projekt abgeschlossen und abgabebereit.

Alternativen

Als Alternative zu IPERKA wird in der Informatikerwelt oft Scrum oder eine modifizierte Version davon verwendet – dies ist eine iterative Art des Projektmanagements: Es wird in einzelnen Sprints immer wieder eine neue Version eines Produktes hergestellt. Dabei wird für jeden Sprint festgelegt, welche Aufgaben gemacht werden, wer was macht usw. Es gibt ausserdem verschiedene Rollen wie den Scrum Master, den Product Owner oder den Entwickler, welche jeweils verschiedene Teile des Prozesses überprüfen und implementieren.

Entscheid

Der Kandidat hat sich gegen solche iterativen Methoden entschieden, da dies aufgrund der vielen nötigen Personen unnötig kompliziert werden würde – ausserdem ist aufgrund der kurzen Zeit von 10 Arbeitstagen die Umsetzung schlicht zu aufwändig im Vergleich mit dem Ertrag.

Es müssten sehr kurze Sprints angesetzt werden von circa etwa 3 Tagen, was zu Stress führen könnte. Ausserdem würde man dafür zusätzlich einen Product Owner und einen Scrum Master benötigen – das könnte zwar unter Umständen der VF übernehmen, aber dieser muss nebenbei selbst auch arbeiten und ist nicht immer erreichbar. Ausserdem würde sich die Arbeit, welche bloss für die Bewältigung eines Sprints nötig wäre, schnell aufsummieren, es müsste am Ende immer eine Besprechung stattfinden, welche für den nächsten Sprint wieder die neuen Aufgaben festlegt. Die Zeit, die dafür verwendet werden würde, könnte man stattdessen für die eigentliche Dokumentation oder das Realisieren verwenden und somit besser nutzen. Ausserdem könnten selbst bei der Verwendung nur sehr wenige und wie erwähnt kurze Sprints durchgeführt werden, was wenig Sinn macht aufgrund der Arbeitslast.

IPERKA ist geeigneter für die kurze Zeit, da bei dieser Methode keine weiteren Personen involviert sind und auch keine Besprechungen stattfinden – weswegen mehr Zeit übrig bleibt und weniger koordiniert werden muss. Ausserdem ist der Kandidat gut mit IPERKA vertraut, da dies schon in diversen Projekten in der Berufsschule erfolgreich angewendet wurde.

Aufbau der Dokumentation

Die Dokumentation richtet sich nach den Vorgaben des Dokuments «Wegleitung und Weisung (FArbeit.pdf)» von der ICT-Berufsbildung Aargau. Sie ist hier abrufbar:
https://www.ict-bbag.ch/wp-content/uploads/2024/11/IPA-Leitfaden_FArbeit_INF_2025-1.pdf

Gemäss den darin enthaltenen Vorgaben ist der IPA-Bericht in zwei Teile geteilt. Der erste Teil bezieht sich auf die Projektstruktur, Umgebung und Aufgabenstellung. Der zweite Teil dokumentiert die eigentliche Arbeit und ist als Folge der Projektmanagement gemäss den Phasen von IPERKA gegliedert.

1.3 Zeitplanung

Bezeichnung	Datum
Start der Arbeit	03.03.2025
Erster Besuch (via Zoom)	05.03.2025 / 14:00
Zweiter Besuch	18.03.2025 / 9:00
Abgabe	24.03.2025 / 13:00
Präsentation und Fachgespräch	09.04.2025 / 14:00

Tabelle 5: Zeitplanung

1.3.1 Arbeitstage

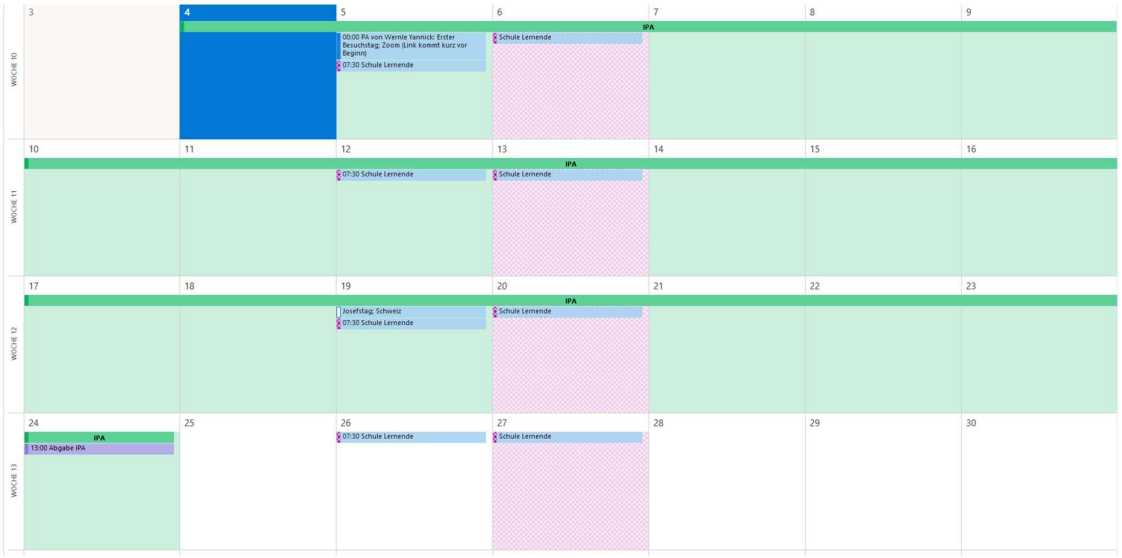


Abbildung 5: Arbeitstage

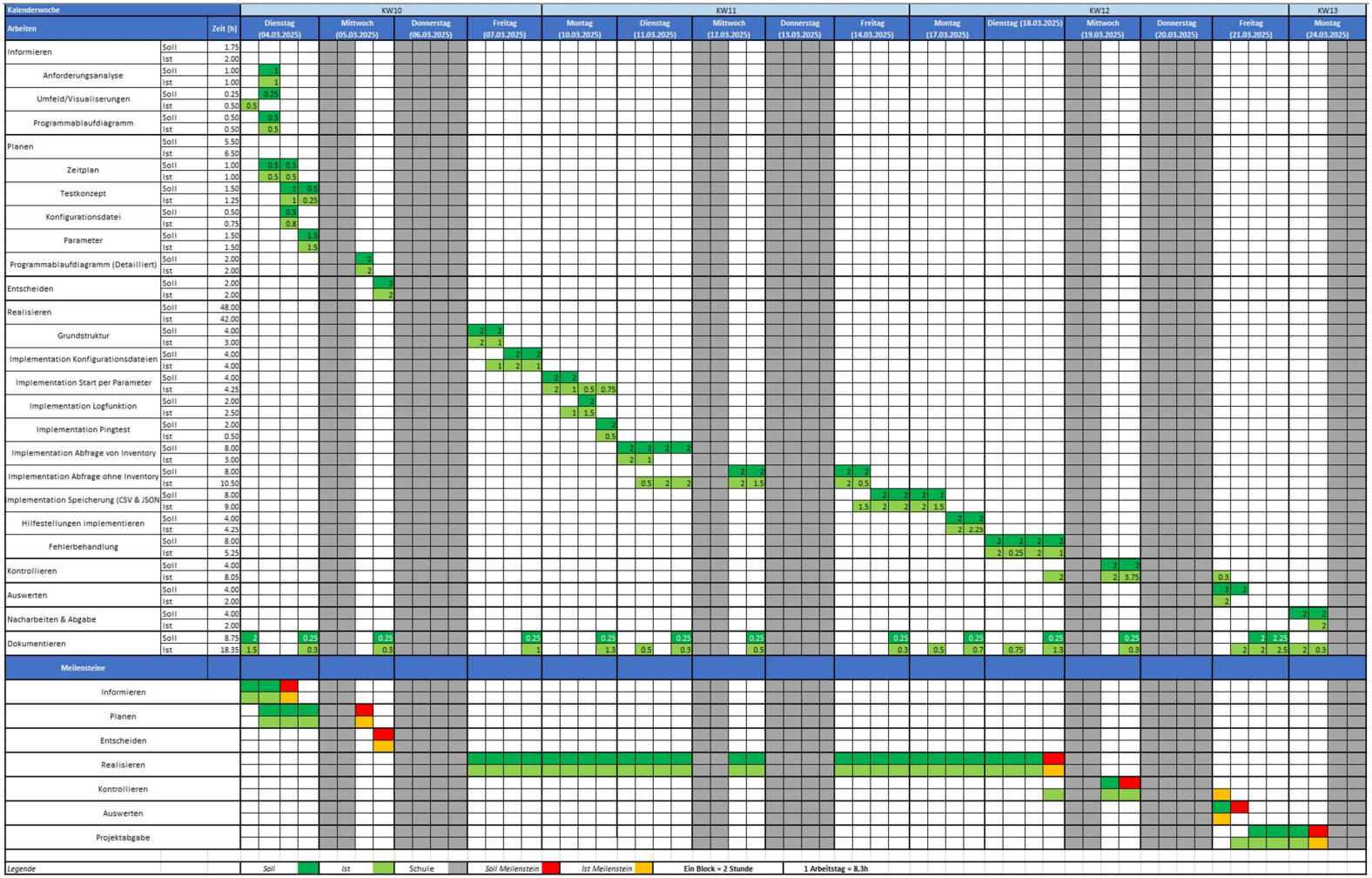
Die Arbeitstage am PSI bestehen aus 8,3 Stunden pro Tag, weswegen ich im Zeitplan und Arbeitsjournal, weswegen auch im Arbeitsjournal immer damit gerechnet wurde. Im Zeitplan wurde das meist weggelassen, da es dort aufgrund der Struktur in 2h-Blöcken schnell zu Verwirrung kommt.

1.4 Meilensteine

Zum Projektmanagement gehören auch Meilensteine – diese resultieren einerseits aus den Phasenübergängen von IPERKA, andererseits aber auch aus den Daten der praktischen Arbeit selbst.

Ende der Informierungsphase
Alle notwendigen Informationen zur Durchführung des Projektes sind erfasst, die Anforderungsanalyse ist fertigstellt.
Ende der Planungsphase
Alles, was im Vorherein planbar ist, ist geplant, um die Umsetzung später im Projekt zu vereinfachen. Das sind unter anderem der Zeitplan, der Programmablaufplan und die Testfälle.
Ende der Entscheidungsphase
Allfällige Entscheidungen, die während der beiden Phasen vorher aufgekommen sind, werden systematisch gefällt, sodass das Projekt durchgeführt werden kann.
Ende der Realisierungsphase
Das Projekt wird umgesetzt und ist im bestmöglichen Zustand. Idealerweise konnten alle Anforderungen implementiert werden.
Ende der Kontrollphase
Das Produkt der Realisation wird auf Herz und Nieren anhand der geplanten Tests und Anforderungen geprüft, um etwaige Verbesserungen zu erfassen.
Ende der Auswertungsphase
Das gesamte Projekt und Produkt werden angeschaut und reflektiert, wo es Verbesserungsmöglichkeiten gibt und was in weiteren Durchgängen noch gemacht werden könnte.
Ende der IPA & Abgabe
Dies ist der Zeitpunkt, in dem die Arbeit in dem Stand, in welchem sie ist, abgegeben und anhand derer die Bewertung durchgeführt wird.

1.5 Zeitplan


KW10
KW11
KW12
KW13
Arbeiten
Zeit [h]
Dienstag (04.03.2025)
Mittwoch (05.03.2025)
Donnerstag (06.03.2025)
Freitag (07.03.2025)
Montag (10.03.2025)
Dienstag (11.03.2025)
Mittwoch (12.03.2025)
Donnerstag (13.03.2025)
Freitag (14.03.2025)
Montag (17.03.2025)
Dienstag (18.03.2025)
Mittwoch (19.03.2025)
Donnerstag (20.03.2025)
Freitag (21.03.2025)
Montag (24.03.2025)
Informieren
Anforderungsanalyse
Umfeld/Visualisierungen
Programmablaufdiagramm
Planen
Zeitplan
Testkonzept
Konfigurationsdatei
Parameter
Programmablaufdiagramm (Detailliert)
Entscheiden
Realisieren
Grundstruktur
Implementation Konfigurationsdateien
Implementation Start per Parameter
Implementation Logfunktion
Implementation Pingtest
Implementation Abfrage von Inventory
Implementation Abfrage ohne Inventory
Implementation Speicherung (CSV & JSON)
Hilfestellungen implementieren
Fehlerbehandlung
Kontrollieren
Auswerten
Nacharbeiten & Abgabe
Dokumentieren
Meilensteine
Informieren
Planen
Entscheiden
Realisieren
Kontrollieren
Auswerten
Projektabgabe
Legende
Soll
Ist
Schule
Soll Meilenstein
Ist Meilenstein
Ein Block = 2 Stunde
1 Arbeitstag = 8,3h

1.6 Arbeitsjournal

Dienstag, 04.03.2025 – Tag 1

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Übernahme der Daten aus PkOrg in Dokumentation o Arbeitsjournal o Grundstruktur o Git-Setup o Teil 1 ausfülle	2h	- Übernahme der Daten aus PkOrg in Dokumentation o Grundstruktur o Git-Setup o Teil 1 ausfüllen	1,5 h
- Informieren o Umfeld o Programmablauf-plan o Anforderungsana-lyse	1,8 h	- Informieren o Umfeld o Programmablaufplan o Anforderungsanalyse	2 h
- Planen o Zeitplan o Konfigurationsda-tei	4,5 h	- Planen o Zeitplan o Konfigurationsdateien o Testkonzept, o Parameterkonzept	4,5 h
		- Arbeitsjournal	0.3h
Überzeit - Soll	0h	Überzeit - Ist	0h
Arbeitszeit - Soll	8,3h	Arbeitszeit - Ist	8,3h

Fazit & Reflexion:

Ich bin heute sehr gut vorwärtsgekommen – ich konnte sehr schnell die Grundstruktur meiner Dokumentation erstellen und die Daten von PkOrg übernehmen und habe auch den grössten Teil des ersten Teils bereits ausfüllen können, sodass ich bereits mit dem Informieren beginnen konnte. Da ich das Umfeld bereits aus der Test-IPA gut kenne, konnte ich auch die Informationsphase, samt Umfeld, PAP und Anforderungsanalyse gut und zackig bereits abschliessen.

Als weiteres habe ich den Zeitplan erstellt – dieser basiert grösstenteils auf der Anforderungsanalyse, da ich anhand derer genauere Vorhersagen machen konnte, welche Funktionen implementiert werden können. Ich habe die Phasen meist so klein wie möglich aufgeteilt, damit ich schnell sehe, wenn ich aus dem Zeitplan gerate und so schnell wie möglich darauf reagieren kann.
Ich bin im Laufe des Tages so weit gekommen, dass ich mit der Planung beginnen konnte – die Konfigurationsdateien, Parameter und das Testkonzept stehen bereits grösstenteils und ich kann morgen mit dem erweiterten Programmablaufplan weiterfahren.

Ich bin für den ersten Tag sehr zufrieden – ich konnte alles abschliessen, was ich mir vorgenommen habe.

Recherchen:

Ist es möglich, Unit-Tests in Powershell zu programmieren? Ja -> (Pester, kein Datum)

Tabelle 6: Arbeitsjournal Tag 1

Mittwoch, 05.03.2025 – Tag 2

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist

- Planen
  o Detaillierter Programmablaufplan
- Erstes Gespräch mit HEX
- Entscheiden
  o Modulansatz oder Scripting
- Arbeitsjournal
	2h
	
	??
	2h
	
	0,25h
		- Planen
		  o Detaillierter Programmablaufplan
		- Erstes Gespräch mit HEX
		- Entscheiden
		  o Kurze Recherche Modulansatz oder Scriptansatz
		  o Entscheidungsmatrix
		- Arbeitsjournal
		- Git eingerichtet
			
			1h
			1h
			2h
			
			0,25h
			0,25h

Überzeit – Soll	0h	Überzeit - Ist	0,25h
Arbeitszeit – Soll	4,25h	Arbeitszeit -Ist	4,5h

Fragen Kandidat Yannick Wernle:	Antworten Fachkraft Helge Brands
Ich habe mich dazu entschlossen, das Programm mit einer Modulstruktur umzusetzen, ist das in Ordnung?	Ja, das ist okay.

Fazit & Reflexion:

Ich habe heute mit der Planung weitergemacht – heute stand vor allem ein Programmablaufplan auf dem Plan, den ich anhand des bisherigen gemacht habe. Ich habe noch einen Zweiten gemacht, da ich mit dem ersten nur eine grobe Übersicht ermöglichen will, aber mit dem Zweiten wollte ich eine grobe Blaupause haben, nach der ich das Programm erstellen kann.

Das erste Gespräch mit dem Hauptexperten war ebenfalls heute und ich denke es ist gut gelaufen – der zweite Termin ist am 18. März geplant. Der HEX hat mir ein paar Hilfreiche Tipps gegeben, die ich sonst vergessen hätte (z.B., dass das Glossar alphabetisch sortiert werden muss usw.).

Ich hatte leichte Schwierigkeiten ein Thema zu finden, zu dem ich eine Entscheidung durchführen kann, mir ist dann aber doch noch eines eingefallen, nämlich ob ich das Script als Modul oder in Scriptform entwickeln soll – die Anwendung würde sich nur geringfügig ändern, aber die Implementation wäre anders – der Modulansatz hat vor allem in der Installation und Auslieferung eine gute Eigenschaften, die ein lineares Script halt einfach nicht kann, ich habe aber mit Modulen bisher wenig bis gar nicht gearbeitet. Ich habe mich aufgrund der Entscheidungsmatrix aber am Ende trotzdem für den modularen Ansatz entschieden.

Ich hatte noch etwas Zeit über und habe schon einmal für Donnerstag Git aktualisiert und mit dem Repository verbunden, welches ich gestern schon erstellt hatte.
Als nächstes steht am Donnerstag die Implementierung der Grundstruktur an und die Implementation der Parameter. Bevor das Losgeht muss ich aber noch die Issues in Github erstellen.

Recherchen:

Recherche, ob eher Modulansatz oder Scriptansatz verwendet werden sollte:
> (Microsoft Learn, 2021) (Oliver, 2011) ([deleted], 2022)

Tabelle 7: Arbeitsjournal Tag 2

Freitag, 07.03.2025 – Tag 3

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Realisierung		- Realisierung	
○ Grundstruktur	3h	○ Grundstruktur	2h
○ Konfigurationsdateien	4h	○ Konfigurationsdateien generieren	4,5h
- Issues in Github			1h
- Arbeitsjournal	1h	- Issues in Github	0,8 h
	0,3 h	- Dokumentieren und Arbeitsjournal	
Überzeit – Soll	0h	Überzeit - Ist	0,2h
Arbeitszeit – Soll	8,3h	Arbeitszeit - Ist	8,5h

Probleme:	Problemlösungen:
/?, --help usw. werden auch ohne Anführungszeichen als Auslöser für die Hilfe akzeptiert, -h aber nicht (wird als Parameter erkannt)	-h als [switch] definieren und die Hilfe dadurch manuell auslösen

Fragen Kandidat Yannick Wernle:	Antworten Fachkraft Helge Brands
Soll die Modulversionsüberprüfung eine spezifische Version oder Mindestversion voraussetzen?	Nein, es soll bei einer anderen Version nur der Hinweis kommen, dass eine andere Version verwendet wurde und es Probleme geben könnte. Am besten wäre auch ein Verweis, wie man die neue Version installieren kann.

Fazit & Reflexion:

Ich hatte für den Morgen eigentlich nur das Implementieren der Grundstruktur und der Issues geplant, ich bin aber bei der Grundstruktur bedeutend schneller vorwärtsgekommen, als ich es geplant hatte, weswegen ich genug Zeit hatte, am Morgen schon mit der Implementation mit den Konfigurationsdateien zu beginnen.

Ich hatte bei der Grundstruktur keine grösseren, erwähnenswerten Probleme gehabt. Das einzig erwähnenswerte ist, dass ich Probleme damit hatte, -h als Hilfsparameter auch zu akzeptieren, doch dann ist mir eingefallen, dass es als [switch]-Typ ganz einfach implementiert werden kann.

Bei der Generierung der Konfigurationsdateien bin ich zwar auch gut vorangekommen, leider jedoch nicht ganz so schnell wie ich erwartet hatte – das lag vor allem an vielen kleinen Fehlern und Problemen. Dadurch, dass ich heute Morgen früher als geplant mit der Grundstruktur fertig war, hält sich der Zeitplan also noch die Waage. Ich hatte am Ende des Tages noch immer genug Zeit, um die Dokumentation weiterzuführen und den nächsten Tag zu planen.

Recherchen:

Dokumentation, wie man ein Modul aufsetzt -> (Marquette, 2017)
Requires-Statement in Modulen -> (sdwheeler A. , 2024)
Special Characters in PowerShell -> (sdwheeler, 2024)

Tabelle 8: Arbeitsjournal Tag 3

Montag, 10.03.2025 – Tag 4

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Realisieren		- Realisieren	
o Start per Parameter	4h	o Start per Parameter	4,5h
o Logfunktion	2h	o Logfunktion	2,3h
o Pingtest	2h	o Pingtest	0.5h
o Arbeitsjournal	0.3h	- Dokumentieren	1h
		- Arbeitsjournal	0.3h
Überzeit -Soll	0h	Überzeit - Ist	0.3h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	8,6h

Probleme:	Problemlösungen:
Beim Update-Config einer existierenden Konfigurationsdatei wurden Daten überschrieben, welche nicht überschrieben werden sollten und solche nicht überschrieben, welche überschrieben werden sollten.	Überprüfung vor dem Update, ob es über die Parameter Änderungen gibt und falls nicht wird der Wert aus der Konfiguration stattdessen ausgelesen.

Fazit & Reflexion:

Ich bin heute auch wieder gut vorangekommen und konnte vor dem Mittag schon den Start per Parameter weitgehend abschliessen. Grössere, nennenswerte Probleme gab es keine. Der Start per Parameter konnte ich unter anderem deswegen schnell abschliessen, da ich mich dazu entschlossen hatte, im Hintergrund eine Konfigurationsdatei aus den Parametern zu generieren – dadurch kann ich bei der weiteren Implementierung sehr viel Zeit sparen, da ich über die Konfigurationsdatei in jeder Situation an die relevanten Daten komme.

Die Logfunktion an sich konnte ich schnell abschliessen, jedoch musste ich am Nachmittag einen Teil der Start-Per-Parameter-Funktion nochmals umschreiben – es gibt dort eine Update-Config-Funktion, welche anhand der Parameter die Konfiguration ausfüllt – ich hatte Probleme beim Umschreiben davon, sodass sie auch bei der Verwendung von einer Konfiguration dazu verwendet werden könnten, einzelne Parameter zu ändern (z.B. nur Loglevel ändern, der ganze Rest soll aus der Konfigurationsdatei kommen). Ich war da etwas langsamer und deswegen hinter dem Zeitplan.

Ich konnte den Zeitplan aber wieder gut aufholen, da ich für den Pingtest nur ca. eine halbe Stunde benötigt hatte anstatt von 2. Ich hatte bei den ersten Implementierungsschritten extra teilweise für auch für die *einfacheren* Aufgaben jeweils 2 Stunden eingeplant, um eben genau solche Schwankungen ausgleichen zu können. Ich bin also wieder im Zeitplan und konnte die restliche Zeit fürs Dokumentieren nutzen.

Recherchen:

Approved Verbs für PowerShell Funktionen -> (sdwheeler c.-h. T.-S., 2024)
Überprüfung ob [switch] verwendet wurde -> (JaapBrasser, 2019)

Tabelle 9: Arbeitsjournal Tag 4

Dienstag, 11.03.2025 – Tag 5

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist

- Implementation
  o Abfrage Inventory
- Arbeitsjournal
	
	8h
	0.3h
		- Implementation
		  o Abfrage Inventory
		  o Abfrage von ILO
		- Arbeitsjournal
		- Dokumentation
			
			3 h
			4,5h
			0.3 h
			0.5 h

Überzeit – Soll	0h	Überzeit - Ist	0h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	8.3h

Fazit & Reflexion:

Ich bin heute weitaus schneller und besser vorangekommen als eingeplant – ich habe anstatt der 8h für die Implementierung der Inventory-Abfrage nur ca. 3h gebraucht, da sie weitaus weniger komplex war als eigentlich gedacht. Ich konnte daher schon am späten Morgen und am Nachmittag mit der Abfrage ohne Inventory beginnen, also der eigentlichen ILO-Abfrage. Wenn ich in diesem Tempo weitermache, dann habe ich gegen Ende der IPA vermutlich noch einige Zeit über – diese könnte ich gut einerseits für die Dokumentation verwenden und ich könnte, falls ich Zeit habe, Unittests doch noch implementieren.

Am Nachmittag bin ich gut vorwärtsgekommen, ich konnte aber die Abfrage von ILO noch nicht abschliessend, da es für die verschiedenen Strukturen in den ILO-Versionen teils sehr (zeit-)aufwändig ist, diese auf den gleichen Stand zu bringen. Bei den Teilen, die ich heute implementiert habe, sind jeweils ILO 4 & 6 abgedeckt – ich muss aber noch testen, inwiefern sich ILO 5 und die anderen Server verhalten. Da ich aber ja mehr Zeit habe wegen meines Tempos heute Morgen, sollte dies hoffentlich keine grösseren Einflüsse auf den Zeitplan haben.

Viel mehr gibt es nicht zu erwähnen, da nicht viel Erwähnenswertes passiert ist, sondern einfach die Arbeit zwar sehr zeitaufwändig, aber gleichzeitig simpel war.

Recherchen:

Select-Object in verschachtelten Arrays -> (codeConcussion, 2013)
Regular Expressions in Powershell -> (sdwheeler, ArieHein, sethvs, Blake-Madden, 2025)

Hilfestellungen:

Regex-Visualisierung für PowerShell -> (gskinner, kein Datum)

Tabelle 10: Arbeitsjournal Tag 5

Mittwoch, 12.03.2025 – Tag 6

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist

- Implementation
  o Abfrage von ILO
- Arbeitsjournal
	4h
	
	0.3h
		- Implementation
		  o Abfrage von ILO
		  o Einbau MAC-Adresse- und Seriennummern-Switch
		  o Scriptmessages implementiert
		- Arbeitsjournal
		- Dokumentieren ILO-Abfrage
			
			2,5h
			0.5h
			
			0.5h
			0.3h
			0.5h

Überzeit – Soll	0h	Überzeit - Ist	0h
Arbeitszeit – Soll	4.3h	Arbeitszeit - Ist	4.3h

Fazit & Reflexion:

Heute war ich vor allem damit beschäftigt, die Abfrage von ILO weiter auszuarbeiten, wobei ich die in Inventory abgelegten MAC 1 bis MAC 4 und die IPv4 sowie IPv6-Konfigurationen dazu gehören. Im Laufe des Nachmittags konnte ich einen grossen Teil der ILO-Abfrage bereits abschliessen, bis auf die Abfrage der in den Servern eingebauten Storage – für die vollständige Abfrage derer müssen die Server laufen, was ich im vollen Büro vermeiden wollte und ich am Freitag in Ruhe umsetzen kann.

Im Laufe des Nachmittages ist mir aufgefallen, dass ich in der Planung der Konfigurationsdatei das Ausschalten der MAC-Adressen und Seriennummern ausgelassen habe. Diese musste ich daher noch im Nachhinein einfügen.

Da im Script bestimmte Vorgänge mit Zeitstempeln in die Konsole geschrieben werden sollen, habe ich dafür die bereits implementierte Logfunktion modifiziert. Ansonsten ist nichts Nennenswertes passiert, was ich erwähnen könnte.

Ich bin weiterhin vor dem Zeitplan und werde voraussichtlich am Freitag die ILO-Abfrage beenden können und die Speicherung in die CSV- und JSON-Dateien beginnen können.

Recherchen:

Entfernen von 2 Array-Elementen vom Anfang -> (DarkLite1, 2014)

Tabelle 11: Arbeitsjournal Tag 6

Freitag, 14.03.2025 – Tag 7

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist

- Implementieren
  o ILO-Abfrage fertigstellen
  o Speicherung in JSON-Datei
  o Speicherung in CSV-Datei
- Arbeitsjournal
	
	4h
	1h
	3h
	0.3h
		- Implementieren
		  o ILO-Abfrage fertigstellen
		  o Speicherung in JSON-Datei
		  o Speicherung in CSV-Datei
		- Arbeitsjournal
			
			2,5h
			0,5h
			5h
			0.3

Überzeit – Soll	0h	Überzeit - Ist	0h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	8.3h

Probleme:	Problemlösungen:
Kompatibilität einiger Funktionen --> Die Abfrage ist je nach ILO-Version nicht möglich.	Abfangen der ILO-Version und bei einer Inkompatibilität wird entweder eine Funktion verwendet, die die gleichen Werte auf anderem Weg holt, oder wenn nicht möglich die Nachricht, dass es nicht Kompatibel ist, gespeichert wird.
Funktion zur Abfrage der Storage exisitert für ILO 6, sie gibt aber (während der Server läuft) einen Fehler zurück, da es einen Fehler mit dem Typ gibt.	Liegt vermutlich an einer HPE-seitigen Inkompatibilität mit meiner PS-Version (7.5). Die Abfrage wird trotzdem durchgeführt – sie sollte in zukünftigen Versionen gefixt werden.

Fragen Kandidat Yannick Wernle:	Antworten Fachkraft Helge Brands
In der Aufgabe steht, dass das Speichern der Seriennummern und MACAdressen in der CSV-Datei abschaltbar sein muss. Wofür ist das gedacht?	In der Zukunft soll irgendwann Inventory die generierte JSON-Datei einlesen können – die CSV-Dateien sind nur für die manuelle Seite gedacht – später soll das Script als Dienst laufen und dann braucht es die CSV-Dateien nicht mehr.

Fazit & Reflexion:

Heute Morgen war ich vor allem damit beschäftigt, die ILO-Abfrage fertigzustellen – für die Abfrage der Storage mussten die Server laufen, weswegen ich aufgrund der Startzeiten der Server teils lange brauchte. Ich bin ausserdem auf ein Problem der Kompatibilität bei der Storageabfrage auf ILO6 gestossen, dass oben beschrieben ist. Die Lösung ist zwar weniger schön, aber da dieser Fehler nicht von meiner Seite gelöst werden kann, sind mir die Hände gebunden. Ich habe hierzu auch Chatgpt gefragt, wie der Fehler sich beheben liesse und er kommt auf eine Inkompatibilität zwischen ILO 6 und der Modulversion.

Den Rest des Tages habe ich damit verbracht, die JSON- und CSV-Dateien abzuspeichern: Die generierung der JSON-Datei ging sehr schnell, da ich die bereinigte Abfrage von ILO mit einem einfachen Befehl konvertieren konnte und ich das bereits bei der Konfiguration benutze.

Deutlich länger dauerte es, die CSV-Dateien zu erstellen: CSV-Dateien sind mehr oder weniger Tabellen, in denen sich die verschachtelte Natur der Abfrage kaum gescheit abbilden lässt. Deswegen brauchte ich etwas länger, unter anderem auch deswegen, weil ich mich dazu entschlossen hatte, für die genauen MAC-Addressen und Seriennummern je eigene Dateien zu erstellen – hierfür musste ich wieder im Hintergrund filtern, was relativ zeitaufwändig war. Ich bin aber trotzdem

noch im Zeitplan und werde voraussichtlich mit der eigentlichen Aufgabe bald fertig, es geht am Montag und die nächste Woche vor allem darum, die Hilfestellungen einzufügen und Fehler auszumerzen.

Recherchen:

Fehler: [...] violates the constraint of type 'T' --> (pl5, 2024)
Überprüfung, ob ein Pfad in einem Directory endet --> (Scripto, 2014)

KI-Prompts:

Wie liesse sich dieser Fehler von HPEILOCmdlets (vers. 4.4.0.0), der Server hat ILO6 beheben?

Tabelle 12: Arbeitsjournal Tag 7

Montag, 17.03.2025 – Tag 8

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Realisieren Abschluss Dateien	3,5h	- Realisieren Abschluss Dateien	3.5h
- Dokumentierung Dateien	1h	- Dokumentierung Dateien	0.5h
- Hilfestellungen	3,5h	- Hilfestellungen	3.75h
- Arbeitsjournal	0.3h	- Fix Update-Config wenn ConfigPath ändert.	0.5h
		- Arbeitsjournal	0.7h
Überzeit – Soll	0h	Überzeit - Ist	0.55h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	8.85

Probleme:	Problemlösungen:
Export-CSV --> Object Reference not set to an instance of an object	Die betroffene Eigenschaft hatte den Wert null bzw. leer --> wird in einem solchen Fall mit «-» ersetzt. Habe dabei auch Chatgpt konsultiert (sinngemäss «hat bei MAC & Generell funktioniert, wieso hier nicht?»)
Wenn das CSV gespeichert wird und das eine Objekt im Array mehr Eigenschaften hat als das andere, wird auf die gemeinsamen Eigenschaften gekürzt.	Loop durch alle einzigartigen Eigenschaften, füge hinzu, falls es nicht bereits existiert (wenn neu hinzugefügt wird, wird die Value «-» verwendet, um zu kennzeichnen, dass es keinen Wert gibt)
Get-Help hat nach der Umschreibung (habe es genereller gemacht) den Fehler geworfen «cannot bind argument to parameter 'helpString' because it is an empty string'	Wird geworfen, wenn ein string-Feld (Parameter) als Mandatory gekennzeichnet wurde aber beim Aufruf dann ein leerer String ist --> Mandatory entfernt und es wird nun in der Funktion auf die Länge überprüft, ob sie länger als 0 ist.
Beim Abspeichern der neuen Configdatei nach dem Update kommt der Fehler «Unable to clear content of 'C:\Users\wernle_y\AppData\Roaming\hpeilo\out' because it is a directory. »	Liegt daran, dass der eingegebene Pfad nur auf einen Ordner deutete und nicht auf wie erwartet auf eine Datei --> wurde korrigiert, sodass der Pfad immer auf eine config.json zeigt.

Fragen Kandidat Yannick Wernle:	Antworten Fachkraft Helge Brands
(Kurze Diskussion mit Helge) Sollen Fehler in Konfigurationsdateien und an sonstigen Orten so weit als möglich korrigiert werden (im Bezug auf Fehler der Konfiguration)?	Die Fehler sollen vor allem bei der Generierung der Konfiguration abgefangen werden. Im restlichen Teil des Programmes soll der User darauf hingewiesen werden (mit Hilfestellungen), aber fixen muss er sie selbst.

Fazit & Reflexion:

Heute stand vor allem der Abschluss der Ausgabedateien im Vordergrund, speziell die CSV-Dateien – ich konnte am Freitag schon einen grossen Teil abschliessen, für heute blieb aber die letzte CSV-Datei übrig, welche die Seriennummern ausgibt. Probleme gab es vor allem bei der Konvertierung, wenn ein Feld leer war – ich musste diese daher mühsam alle abfangen.

Ich musste ausserdem eine Funktion erstellen, welche die Keys unter allen abgefragten Servern standardisiert – wenn dem nicht so ist, werden nämlich einfach nur die Felder angezeigt, deren Key bei

allen vorhanden ist – das ist nicht schön, weil so einfach Daten verloren gehen. Ausserdem habe ich bei den Seriennummern die Tabelle erweitert, sodass nicht nur die Seriennummern angezeigt werden, sondern in einer separaten Tabelle auch Details dazu, damit man weiss, was zu was gehört.

Helge und ich haben kurz vor dem Mittag noch darüber geredet, wie der Stand ist und haben uns dann entschlossen, die IPA testweise am Mittwochnachmittag auf den SwissFEL-Maschinen zu testen.

Am Nachmittag habe ich mich vor allem mit den Hilfestellungen befasst und sowohl für das Modul als auch für alle Public-Funktionen Comment-Based Hilfestellungen einfügt. Ich habe dann auch noch ca. eine halbe Stunde dafür aufgewendet, das beim Editieren des Konfigpfads in der Funktion Update-Config die Konfiguration wirklich auch dorthin verschoben wird. Im Nachhinein ist das eher unnötig gewesen, da es zwar «nett» ist, aber weder in den Anforderungen ist noch grosse Relevanz besitzt. Ich hätte es besser in neue Testfälle und Fehlerabfangen nützen können, das hätte mehr gebracht.

Ich habe ausserdem bei der Planung ein wenig getrickst, denn im Nachhinein ist mir aufgefallen, dass die beiden Teile «Hilfestellungen» und «Fehlerbehandlung» sich sehr viel überschneiden – inhaltlich vor allem bei den Hilfestellungen bezgl. von Fehlern in den Konfigurationsdateien oder Fehler bei den Parametern. Was diese Hilfe angeht, habe ich morgen zwar den gesamten Tag Zeit dafür, jedoch weiss ich nicht, ob ich trotz des zweiten Besuches genug Zeit habe, um das zufriedenstellend zu machen – ich habe zwar schon jede Menge abgefangen bei der Implementation, aber da muss ich noch an einigen Stellen entwirren und vor allem muss ich überprüfen, ob die verwendeten Parameter auch jeweils validiert werden.

Ich musste heute auch ein wenig Überzeit einlegen, da ich nicht vor den 8.3h mit dem Arbeitsjournal und dem Drumherum fertig geworden bin.

Recherchen:

Export-CSV Object reference Problem --> (jeremy.hawkins1, 2023)
Woher kommt eine Ausgabe der Konsole --> (Maclean, 2018)
Commentbased Help --> (sdwheeler A. , 2025)

KI-Prompts:

- Sinngemäss: Wie mache ich einen Abstand zwischen zwei separaten Teilen innerhalb einer CSV-Datei?
- Sinngemäss: Ich habe in einer Funktion “Cannot bind argument to parameter 'helpString' because it is an empty string” erhalten. Woran liegt das?
- Sinngemäss: Woher kommt der Fehler «Unable to Clear Content of [...] because it is a directory” bei der Verwendung von Set-Content?

Tabelle 13: Arbeitsjournal Tag 8

Dienstag, 18.03.2025 – Tag 9

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist

- Fehlerbehandlung
  o Validierung der Konfiguration, Parameter usw.
  o Fehler abfangen.
- 2er Expertenbesuch
- Testfälle überarbeiten
- Arbeitsjournal
	5h
	
	1h
	2h
	0.3h
		- Fehlerbehandlung
		- Fix Securestring
		- Expertenbesuch
		- Testfälle überarbeiten
		- Arbeitsjournal & Nachführung Dokumentation
			4,5h
			0.75h
			0.75h
			2h
			1.3h

Überzeit – Soll	0h	Überzeit - Ist	1h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	9.3h

Fazit & Reflexion:

Am Morgen habe ich heute vor allem bei der Fehlerbehandlung gearbeitet: D.h. ich habe vor try-catches eingebaut und die vorhersehbarsten Fälle abgefangen. Ausserdem war das zweite Gespräch mit dem Expert, welches wie ich finde gut gelaufen ist – Helge und der HEX schienen ziemlich zufrieden zu sein mit meinem Code und meiner Dokumentation und haben keine grösseren Probleme und Schwachstellen finden können. Weiteres dazu im Protokoll im Anhang (10.2). Das Gespräch habe ich im Zeitplan heute als Dokumentierung gekennzeichnet (da es doch 1h gedauert hat.

Die Fehlerbehandlung selbst war teils umständlich, da ich alles von Hand testen musste und es schwer war, alle verschiedenen Fälle zu berücksichtigen. Aus diesem Grund habe ich mit dazu entschlossen, am Nachmittag meine Testfälle zu überarbeiten – in der Planung waren dies noch wenig und allgemein gehalten, da ich nicht genau abschätzen konnte, was ich wie implementieren würde. Jetzt am Ende des Projektes kann ich das relativ genau abschätzen und habe sie deswegen angepasst – ausserdem konnte ich so besser die Fehlerbehandlung testen und Details hinzufügen.

Ich war den ganzen Nachmittag abwechselnd damit beschäftigt, Fehler zu finden und sie zu beheben, parallel die Testfälle zu überarbeiten und weitere zu schreiben. Ich habe währenddessen unter anderem auch alle Wege noch mal durchprobiert, wie das Programm gestartet werden kann. Ich wäre im Nachhinein sehr, sehr froh darüber gewesen, wenn ich hierfür Unittests hätte, denn dann hätte ich nicht jedes Mal mühsam alles händisch testen müssen, was sehr mühsam und zeitaufwendig war. Ich wurde nicht rechtzeitig mit meinen geplanten Arbeiten fertig, weswegen ich Überzeit machen musste, um das alles noch zu schaffen.

Vermutlich muss ich morgen ebenfalls Überzeit machen, da ich die Logs und Kommentare sowie das Modulmanifest noch nachführen möchte.

Beim Überprüfen der Testfälle mit den Anforderungen ist mir dann noch aufgefallen, dass ich den Parameter um den Pingtest ausschalten übersehen hatte und nicht implementiert war, was ich dann auch schnell noch nachkorrigiert hatte.

Ich konnte trotzdem die Implementationsphase abschliessen, da jetzt alles laufen sollte.

Recherchen:

Konvertierung zu SecureString --> (Microsoft, kein Datum)
-Contains funktioniert nicht wie gedacht --> (tnw, 2013)

Hilfestellungen:

Kleine Anmerkungen beim zweiten Expertengespräch:

- SecureString sollte anstatt String bei Passwörtern verwendet werden

KI-Prompts:

Sinngemäss: Ich möchte für ein cmdlet in powershell ausschalten, dass der rückgabewert angezeigt wird. Wie mache ich das?
Sinngemäss: Wie kann ich bei Powershell SwitchCase mehrere Möglichkeiten dasselbe machen lassen?

Tabelle 14: Arbeitsjournal Tag 9

Mittwoch, 18.03.2025– Tag 10

Geplante Arbeiten:	**Soll**	**Erledigte Arbeiten:**	**Ist**

- Testing/Kontrollieren
  o Testing auf Scharfer Maschine
  o Testing gemäss Testfällen
  o Fixen kleinerer Probleme
- Arbeitsjournal
	
	1h
	2h
	1h
	0.3h
		- Testing/Kontrollieren
		  o Testen auf Scharfer Maschine
		  o Testing gemäss Testfällen
		  o Fixen kleinerer Probleme
		  o Einbau Pfade & kleinere Bugbehebungen
		  o Arbeitsjournal
			
			0.75h
			2,5h
			1,5h
			1h
			0.3h

Überzeit – Soll	0h	Überzeit - Ist	1,75h
Arbeitszeit – Soll	4.3h	Arbeitszeit - Ist	6h

Probleme:	**Problemlösungen:**
Boolwerte in der Konfiguration werden immer wieder überschrieben	Lag an der Implementation der Update-Config-Funktion, welche diese immer zurücksetzte. Das wird umgangen, indem diese Values jedes Mal mitgegeben werden, um die Sicherung zu garantieren

Fazit & Reflexion:

Der heutige Tag war sehr stressig, da ich alle 18 Testfälle durchmachen musste. Nachdem ich sie geschrieben hatte, habe ich begonnen, sie zu testen. Ich hatte aber bei nahezu jedem Test einen Fehler, obwohl ich eigentlich gestern schon versucht hatte, alles möglichst gut zu testen und abzuschliessen. Aus diesem Grund musste ich bei vielen Tests die Fehler noch ausbügeln, um das Script lauffähig zu machen, weswegen ich heute auch Überstunden machen musste.

Zu diesen Fehlern gehörten unter anderem die korrekte Validation der Typen aus der Konfiguration oder ein Fehler beim Update der Konfigurationsdatei. Weitere nennenswerte Fehler kann ich ehrlich gesagt nicht nennen. Lediglich beim Test auf der scharfen Maschine schien das Logging nicht zu funktionieren; Auf meinem PC lokal aber schon. Ich werde deswegen das in den nächsten Tagen noch überprüfen und wenn möglich noch anpassen. Die Kontrollieren-Phase ist somit eigentlich auch schon wieder abgeschlossen.

Beim Test auf der scharfen Maschine, welcher mir Helge vorgeschlagen hatte, sind ausserdem weitere kleine Probleme und Fehler aufgetaucht (wie Generierung einer server.json Datei, aber Pfad auf servers.json usw.). Im Zuge dessen hat er sich auch noch gewünscht, dass ich eine kurze Funktion, welche alle Pfade überprüft und im schlimmsten Fall erstellt. Diese war schnell gemacht, hat aber auch noch Überzeit eingebracht.

Zwischenzeitlich war ich sehr gestresst, weil ich so viele Fehler erhalten hatte, aber am Ende des Tages kann ich sagen, dass zum Glück das meiste so funktioniert wie es sollte und ich mich die nächsten Tage vor allem auf die Dokumentation fokussieren kann. Es hat mich sehr erleichtert, dass am Ende auf der scharfen Maschine alles funktioniert hat und somit ein vorzeigbares Resultat existiert.

Am Ende haben Helge und ich ausserdem das Resultat an Kurt weitergeleitet, um ihn nach Feedback zu fragen, ob alle notwendigen Daten darin enthalten sind usw.

Recherchen:

Recherche, wie Typen in PowerShell geprüft werden können --> (Powell, 2012)

Hilfestellungen:

VF --> Testen auf scharfer Maschine und kurze Unterhaltung über Optimierungspotenzial

Tabelle 15: Arbeitsjournal Tag 10

Freitag, 21.03.2025 – Tag 11

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Refactoring, - Logs nachführen - Logging Testen - Kommentare nachführen - Modulmanifest - Nachführung Dokumentation - Auswerten Phase	1h 0.5h 0.3h 0.5h 0.5h 3.5h 2h	- Refactoring, - Logs nachführen - Logging Testen - Kommentare nachführen - Modulmanifest - Nachführung & Verbesserung Dokumentation - ILO-Beschreibung hinzugefügt - Inventory-Beschreibung hinzugefügt - Auswerten Phase	0.5h 1h 0.3h 0.5h 0.25h 3.25h 0.5h 0.5h 2h
Überzeit – Soll	0h	Überzeit - Ist	0.5h
Arbeitszeit – Soll	8.3h	Arbeitszeit - Ist	8.8h

Fazit & Reflexion:

Als erstes habe ich heute Morgen das Logging getestet, welches auf der Testmaschine nicht funktioniert hat: Ich habe hierfür die Remote-Maschine gfa-cinteg01.psi.ch verwenden können, dass als «leere» Maschine perfekt fürs Testen geeignet war. Ich konnte den Fehler nicht replizieren.

Ansonsten stand heute vor allem noch das Nachführen der Logs, ein kurzes Refactoring an einigen Stellen im Code (Vereinheitlichung der Schreibweise der Variablen), sowie das Ausfüllen des Modulmanifests. Alle diese Dinge konnte ich problemlos am frühen Morgen abschliessen und mit der Auswertungsphase beginnen.

Die Auswertungsphase konnte ich ebenfalls schnell abschliessen – von den im Zeitplan geplanten 4 Stunden waren sowieso etwa 2h davon als Puffer gedacht, sodass ich danach genug Zeit hatte, die Dokumentation noch zu verbessern und zu erweitern. Ich habe zur besseren Erklärung unter anderem im Informierungsteil der Dokumentation einen kurzen Teil zur Erklärung von ILO und Inventory hinzugefügt, um es für aussenstehende einfacher zu machen, die Dokumentation und die Arbeit zu verstehen.

Den Rest des Tages habe ich damit verbracht, die Dokumentation und deren Darstellung zu verbessern.

Recherchen:

Recherche zu ILO --> (Hewlett Packard Enterprise, kein Datum), (Kaarsemaker, 2021), (Wikipedia, 2025), (Wikipedia, 2025)

Tabelle 16: Arbeitsjournal Tag 11

Montag, 24.03.2025 – Tag 12

Geplante Arbeiten:	Soll	Erledigte Arbeiten:	Ist
- Korrigierung Dokumentation	3h	- Korrigierung Dokumentation	2,5h
- Abgabe	1h	- Abgabe	0.5h
- README	0.7h	- README	0.7h
- Arbeitsjournal	0.3h	- Arbeitsjournal	0.3h
Überzeit – Soll	0h	Überzeit - Ist	0h
Arbeitszeit – Soll	5 h	Arbeitszeit - Ist	4h

Fragen Kandidat Yannick Wernle:	Antworten Fachkraft Helge Brands

Fazit & Reflexion:

Am heutigen Tag habe ich mich praktisch nur noch auf die Dokumentation fokussiert: Ich hatte sie über das Wochenende meiner Familie gegeben, um die Grammatik und Rechtschreibung zu prüfen und habe die Änderung heute Morgen hinzugefügt.

Ausserdem habe ich meinem Github-Repository noch ein README hinzugefügt sowie alle Dateien & Abbildungen gezippt. Ich habe ausserdem auch nochmal Kurt Bitterli per Teams, ob er sich bereits die CSV-Dateien vom scharfen Test hat anschauen können. Das hat er und laut ihm sind alle wichtigen Daten darin enthalten – er beginnt jetzt schon damit, anhand dieser Daten Inventory zu aktualisieren.

Tabelle 17: Arbeitsjournal Tag 12

2 Teil 2 – Projektdokumentation

2.1 Kurzfassung des IPA Berichts

2.1.1 Ausgangslage

Am PSI werden an den Grossforschungsanlagen Serversysteme eingesetzt, welche ein Wartungsinterface (Integrated Lights Out) besitzen. Über dieses lassen sich Wartungsaufgaben für den jeweiligen Server ausführen und Informationen über diesen auslesen.

Um einen Überblick über die Anlagen und deren verbauten Server und Komponenten zu haben, ist ein Tool nötig, welches die wichtigsten MAC-Adressen und Seriennummern des Servers ausliest und einen Überblick dazu ausgibt. Es muss in PowerShell geschrieben werden, da eine Bibliothek zur Abfrage bereits in dieser Sprache vorhanden ist. Um die Zielsysteme gezielt abzufragen, sollen anhand eines Suchstring die Server aus der internen Datenbank inventory.psi.ch abgefragt werden.

2.1.2 Vorgehen

Das Projekt wurde in Form eines PowerShell-Moduls implementiert, welches neben der Hauptfunktion weitere Funktionen zur einfacheren Handhabung bereitstellt. Die Abfragen von ILO und Inventory werden auf der Basis von generierten Konfigurationsdateien durchgeführt, welche durch den User generiert und befüllt werden. Es ist ebenso möglich, das Programm ohne Konfiguration mit Parametern zu starten. Es bestehen ausserdem noch Elemente zur Unterstützung des Users in Form von Hilfestellungen.

Herausforderungen gab es vor allem im Bezug auf die Fehlerbehandlung und Hilfestellungen bei Fehlern.

2.1.3 Ergebnis

Das Ergebnis der Arbeit ist ein funktionales PowerShell-Modul, welches bereits im Abgabezustand verwendet werden kann. Es erfüllt die gestellten Anforderungen weitestgehend, es gibt aber auch Elemente (bspw. in der Fehlerbehebung), welche noch Optimierungspotenzial besitzen.

2.2 Informieren

2.2.1 Projektumfeld

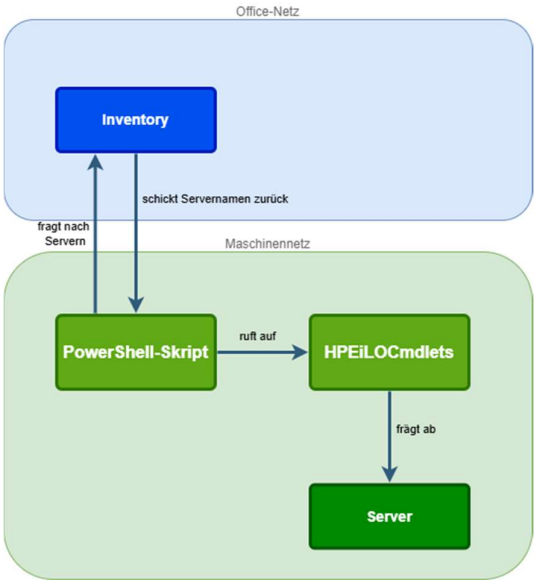


Abbildung 6: Projektumfeld

Das Umfeld des Projekts ist in der Abbildung dargestellt – grundsätzlich läuft es so ab, dass sich die Person, die das Script ausführen will, per Remotedesktopverbindung auf einen Server einloggt. Diese Server befinden sich in einem gesonderten, gesicherten Netz ohne Internetzugang. Da das Script lokal ausgeführt wird, befindet es sich ebenfalls in diesem Netz – um das zu ermöglichen muss die HPEiLOCmdlets-Bibliothek lokal vorhanden sein.

Das PowerShell-Script muss aber gleichzeitig vom Inventory, welches sich im Officenetz (mit Internetzugang) befindet, Daten abfragen – namentlich die Hostnamen fürs iLO. Inventory stellt diese per API bereit.

Als Zielgruppe im weitesten Sinn kann man meinen Fachvorgesetzten nennen – er benötigt das Script, um Inventurdaten zu erfassen. Die einzige andere Person, die noch zur Benützung in Frage käme, ist Kurt Bitterli, welcher die Gruppe leitet, die Inventory grossflächig zur Inventarisierung nutzt. Im Vorfeld der IPA wurde aber entschieden, Kurt nicht in die IPA zu integrieren, da er einerseits genug anderes zu tun hat, und andererseits seine Ansprüche & Vorschläge den Zeitrahmen der IPA sprengen würden.

2.2.2 Systemgrenzen & Schnittstellen

Die Grenzen des Systems bildet vor allem die Abfrage vom Inventory und gleichzeitig von den Servern – es ist als normaler User am PSI nicht möglich, sich gleichzeitig mit zwei verschiedenen Netzen zu verbinden. Das bedeutet, die beiden Teile (Abfrage Inventory und Server) müssen separat voneinander funktionsfähig sein.

Ausserdem ist das Einzige, was als Resultat des Scripts übrigbleiben soll, Dateien mit den ausgelesenen Daten – sie werden weder per Mail verschickt, noch sonst irgendwie bereitgestellt. Sprich, es soll kein Update im Inventory durch das Script ausgelöst werden.

Schnittstellen gibt es vom *normalen* PowerShell zur «HPEiLOCmdlets»-Bibliothek, welche die HP-Server per iLO-Technologie abfragt. Weiter gibt es auch die Schnittstelle von PowerShell zur WebAPI von Inventory – diese kommunizieren über HTTPS miteinander.

Eine sehr wichtige Grenze des Systems ist die Art an Systemen, welche Abgefragt werden können – iLO ist eine Technologie, welche nur von HPE-Servern bereitgestellt werden. Das bedeutet, dass bspw. Netzwerkkarten oder andere im Server eingebaute Teile, die nicht von HPE verwendet oder verkauft werden, nicht abfragbar sind mit einem solchen Script. Ausserdem werden selbstgebaute Teile in der Ausgabe von iLO nicht mit dem gleichen Informationsumfang beschrieben, wie ihre eigenen (Nur «PCIe-Card» statt konkreter Bezeichnung).

In Absprache mit dem VF wurde festgelegt, dass die Implementation des Programmes sich spezifisch auf die in der Aufgabenstellung bezeichneten Servertypen fokussiert wird. Das Programm soll am Ende zuerst auf der SwissFEL-Anlage laufen und beschränkt sich auch dort auf die Kameraserver (gemäss Namenskonvention also z.B. der Server «sf-sioc-cs-64»)

2.2.3 Anforderungsanalyse

In der Anforderungsanalyse werden alle Anforderungen an das Produkt dargestellt und einzeln erfasst. Sie basieren auf der detaillierten Aufgabenstellung und sollen so konkret sein, dass sie sich jeweils mit nur einem Feature befassen.

Es wird zwischen Funktionalen Anforderungen (ANF-XX), qualitativen Anforderungen (ANF-Q-XX) und Rahmenbedingungen (ANF-R-XX) unterschieden. Für die funktionalen Anforderungen gibt es aufgrund der Anzahl eine eigene Tabelle.

Funktionale Anforderungen

Anforderung	Beschreibung
ANF-01	Es wird auf Anforderung des Users eine Hilfeseite angezeigt
ANF-02	Es wird bei fehlerhaften Parametern dem User eine Hilfeseite angezeigt und auf den Fehler hingewiesen.
ANF-03	Es wird bei Fehlern in den Konfigurationsdateien eine Hilfeseite angezeigt und auf den Pfad und das Problem hingewiesen.
ANF-04	Es wird bei der Initialisierung des Programmes die Version der Bibliothek verglichen und auf Unterschiede hingewiesen.
ANF-05	Das Programm soll per Konfigurationsdatei konfiguriert werden können.
ANF-06	Es soll eine leere Konfigurationsdatei generiert werden können.
ANF-07	Es soll eine Konfigurationsdatei generiert werden können, welche Beispieldaten beinhaltet.
ANF-08	Das Programm soll ebenfalls nur per Kommandozeile gestartet werden können.
ANF-09	Es soll eine Logfunktion geben, bei der Aktivierung, Level und Zielort konfiguriert werden kann.
ANF-10	Es soll Parameter geben, welche mittels Suchstring Inventory abfragen und diese zur iLO-Abfrage verwenden.
ANF-11	Es soll Parameter geben, welche anstatt von Inventory eine Liste an Hostnamen zur iLO-Abfrage verwenden

ANF-12	Das Script soll so aufgebaut werden, dass auch ohne Inventory das Script zur Abfrage verwendet werden kann.
ANF-13	Es soll einen Parameter geben, welcher den Pfad zur Konfiguration enthält.
ANF-14	Es soll einen Parameter geben, welcher den Pfad zum Report enthält.
ANF-15	Es soll einen Parameter geben, welcher einen Pfad enthält, wo nach den benötigten Dateien gesucht wird.
ANF-16	Es soll ein Verbindungstest/Pingtest durchgeführt werden, um zu testen, ob der gewünschte Server erreichbar ist.
ANF-17	Es soll einen Parameter geben, welcher den Pingtest ausschalten lässt.
ANF-18	Es sollen vom Server die MAC-Adressen und Seriennummern abgefragt werden können.
ANF-19	Das Ergebnis der Abfrage soll in einem JSON-File gespeichert werden.
ANF-20	Das Ergebnis der Abfrage soll in einem CSV-File gespeichert werden, welches den Hostnamen, die Seriennummer und die Anzahl Netzwerkinterfaces beinhält.
ANF-21	Das Speichern von MAC-Adressen und Seriennummern soll jeweils per Parameter ausschaltbar sein.
ANF-22	Die Ausgaben in der Konsole sollen Zeitstempel beinhalten.
ANF-23	Die Konsole soll den Verbindungsaufbau und Stand der Abfrage auf dem Bildschirm zeigen.
ANF-24	Es sollen Fehler und das Verhalten während der Laufzeit in einem Logfile gespeichert werden.

Nicht Funktionale Anforderungen

Anforderung	Beschreibung
ANF-Q-01	Die Datenstruktur des Reports soll ähnlich zum Inventory sein.
ANF-R-01	Das Script wird mittels PowerShell entwickelt & verwendet die Bibliothek «HPEiLOCmdlets».
ANF-R-02	Das Script soll per Abfragen Daten von den spezifizierten Servern auslesen und abspeichern.
ANF-R-03	Das Script mit den Servern DL380 Gen8/Gen9/Gen10/Gen11& DL20 Gen10 getestet werden.

2.2.4 Programmablaufplan

Aufgrund der sehr kleinen Zielgruppe des Scriptes und als Resultat davon, dass sich keine verschiedenen Rollen aus der Aufgabenstellung ableiten lassen, ist es nicht zielführend hier mit Personas, Use Cases oder User Stories zu arbeiten, da diese mehrere Rollen voraussetzen – sie trotzdem zu machen wäre weder effizient noch notwendig, da die Wünsche der einzigen Rolle schon in der Anforderungsanalyse persönlich abgebildet sind.

Weitaus zielführender ist hingegen das Erstellen eines Programmablaufplans. Durch ihn können die relevanten Abläufe und Schnittstellen besser dokumentiert werden und sind auch für das Verständnis des gesamten Scripts als Aussenstehender wichtig.

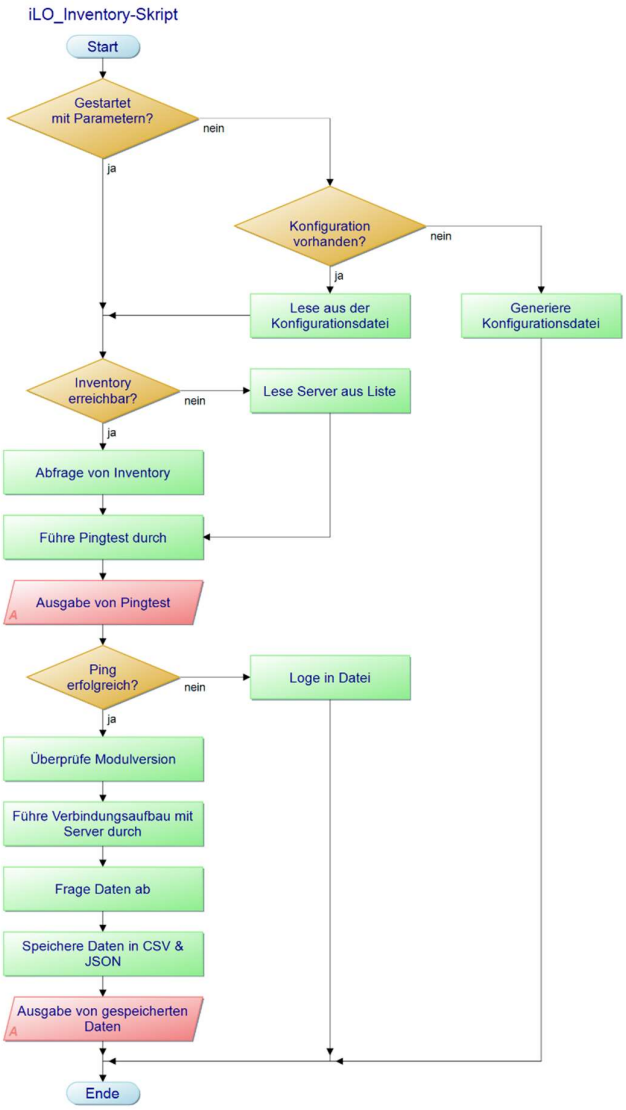


Abbildung 7: Programmablaufplan (simpel)

Dieser PAP ist lediglich als grober Ablauf des Programms zu verstehen – genaue Abfragen und Optionen, wie sie z.B. bei der Aktion «Generiere Konfigurationsdatei» sind, wurden bewusst weggelassen, sodass dieser Plan eine Übersicht aller geplanter und vorgegebenen Funktionen darstellt. In der nächsten Phase, der Planung, gibt es einen weiteren, weitaus detaillierteren Ablaufplan, anhand dessen das Script im Endeffekt implementiert wird.

2.2.5 Integrated Lights Out (ILO)

Integrated **L**ights **O**ut (kurz ILO) ist ein von Hewlett Packard Enterprise (kurz HPE) für ihre ProLiant-Server bereitgestellte Out-of-Band Managementtechnologie. Diese Technologie ermöglicht es, über den ILO-Chip aus der Ferne auf den Server zuzugreifen, in dem er verbaut ist.

Dies geschieht über eine vom eigentlichen Server komplett unabhängige Netzwerkkarte und Netzwerkverbindung (daher auch die Out-of-Band Eigenschaft), über die der Server in jedem Zustand erreicht werden kann, ob dieser läuft, kaputt ist oder komplett abgeschaltet ist.

Abbildung 8: Gesonderte Netzwerkschnittstelle von ILO

ILO stellt über diese Verbindungen verschiedene Funktionen bereit wie bspw. das Ein- und Ausschalten der Server. Mit jeder Version kommen neue Funktionalitäten mit, welche die Informationsspanne erweitern: Mit ILO 6 kann z.B. auch der Speicher abgefragt werden. Ganz generell (und wichtigste Eigenschaft für die IPA) lässt sich über ILO auch Eigenschaften wie die MAC-Adressen oder Seriennummern von den verbauten Komponenten auslesen.

Durch dieses Remotemanagement muss man also nicht immer zum physischen Server, um diesen zu konfigurieren oder Hardwaredaten davon abzulesen, sondern kann dies bequem vom Büro aus machen.

Es existieren verschiedene Programmierschnittstellen, welche die ILO-Funktionen nicht nur per Browser erreichbar machen. Dazu gehört auch neben der verwendeten HPEiLOCmdlets-Bibliothek auch Bibliotheken für Perl, Python oder Ruby.

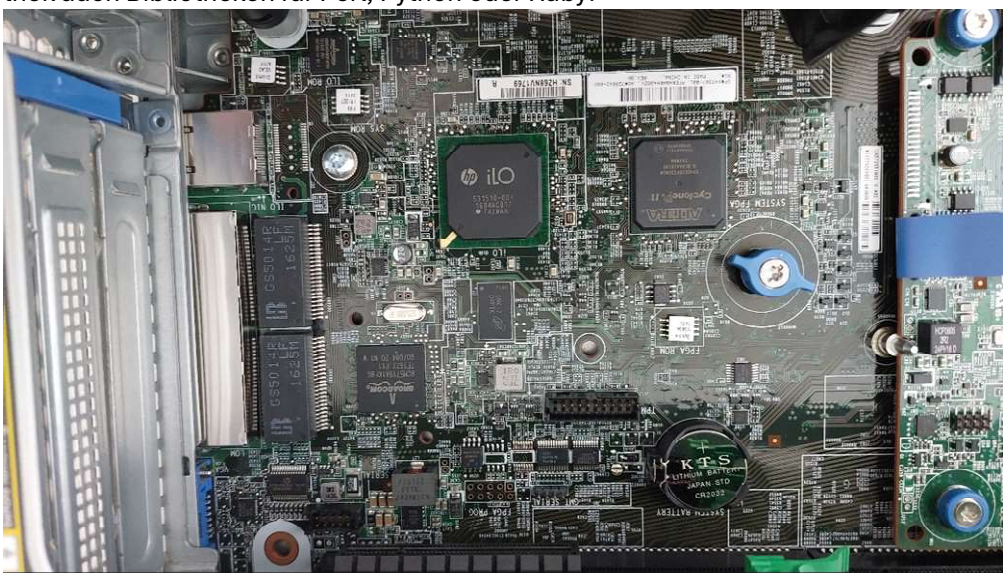


Abbildung 9: ILO-Chip in einem ProLiant Gen9

2.2.6 Inventory.psi.ch

Inventory.psi.ch ist die Datenbank zur Erfassung und Inventarisierung von externen und internen Komponenten. Sie wird primär durch die Gruppe Hardware Support (Lead: Kurt Bitterli) benützt und durch Alain Bertrand gewartet.

Inventory enthält also nicht nur Server und Computer, welche an den Grossforschunganlagen installiert oder an Lager sind, sondern auch Kabel, Netzwerkkarten uvm. Neben den Informationen über welche Teile und Parts existieren speichert Inventory auch ab, wie solche Teile miteinander verknüpft sind. Als zweites Main Feature können über Inventory auch Teile bestellt werden.

Für die IPA ist aber lediglich die Informationsseite, spezifisch die der Server relevant, da diese die relevanten Daten zur ILO-Abfrage enthält. Eine solche Seite kann z.B. so aussehen:

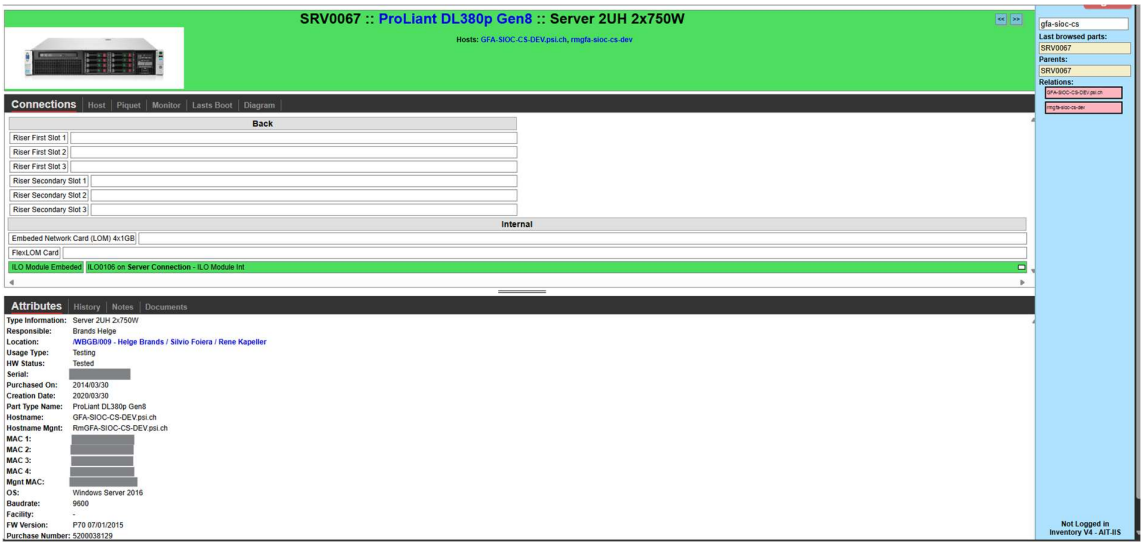


Abbildung 10: Informationsansicht Inventory

Inventory stellt ausserdem eine JSON API bereit, welche alle Daten, die auch auf dieser Seite angezeigt werden, zugänglich macht. Auf dieser Grundlage soll die Inventoryabfrage vom Programm durchgeführt werden. Im Bezug auf ILO ist vor allem das Feld **Hostname Mgnt** wichtig, welches den Hostnamen vom ILO darstellt. Die Felder **MAC 1 – MAC 4** sind jeweils die MAC-Adressen der verbauten Netzwerkkarten, das Feld **MAC Mgnt** ist wiederum die MAC-Adresse der ILO-Netzwerkschnittstelle.

2.3 Planen

2.3.1 Testkonzept

Testsystem

Die Tests, welche anhand der Aufgabenstellung durchgeführt werden müssen, werden in Umgebungen getestet, welche der tatsächlichen Anwendung auch so vorkommen könnten. Dabei gibt es zwei unterschiedliche Fälle: Die Server hängen im Officenetzwerk oder sie hängen im Maschinennetz.

Getestet werden soll, ob das Script wie geplant funktioniert und reagiert, vor allem im Bezug auf die Erreichbarkeit des Inventory, sowie die komplette Funktionstüchtigkeit. Es ist in PowerShell zwar möglich, Unittests mittels Pester zu implementieren, jedoch ist das nicht Bestandteil der Aufgabenstellung – gefordert sind nur Laufzeittests.

Voraussetzungen

Der Laptop des Kandidaten muss angeschaltet sein und sich in einem Netzwerk befinden, in dem sich mind. Einer der 5 Server befindet. Ausserdem muss die richtige PowerShell-Version (mind. v.7) installiert sein. Ausserdem muss der iLO-Server, der abgefragt werden soll, am Strom hängen – sonst kommen keine Daten zurück. Sie müssen, wenn möglich, am Laufen sein, damit die volle Informationsbreite getestet werden kann (wenn der Server nicht läuft und nur das iLO geben manche der Befehle nicht alle Daten zurück.

Testumgebung

Jegliche Tests wurden auf dem persönlichen PSI-Laptop des Kandidaten durchgeführt. Dieser hat die folgenden Spezifikationen:

PSI Laptop	PC14734 / HP EliteBook 840 G7 Notebook PC
Betriebssystem	Windows 11 Enterprise (23H2) – Build 22631.4317
Powershell	Version 7.5.0
VS-Code	Version 1.97.0
VS-Code PowerShell Extension	2025.0.0

Testmittel und Methoden

Es werden **Laufzeittests** durchgeführt – das heisst, sie werden getestet, während das Programm effektiv so läuft wie es nach der Auslieferung auch laufen würde.

User Tests

Die Testfälle werden in Form von User Tests getestet, sprich sie werden von Hand und von einer echten Person getestet. Es werden dafür Testfälle geschrieben, welche die genaue Vorgehensweise beim jeweiligen Test angeben. Zu jeder Testfallspezifikation gibt es einen Eintrag im Testprotokoll. Im Testbericht ist das Ergebnis und falls vorhanden, notwendige Verbesserungen festgehalten.

Nicht getestete Aspekte

Als Folge der Durchführung als User Tests wird nur das Zusammenspiel der Funktionen als Endprodukt getestet, nicht aber alle Funktionen, die es im Script gibt. Diese liessen sich in Zukunft mit Unittests abdecken.

2.3.2 Testfälle

Testfall:	**TF-01**	**Anforderung:**	**ANF-01**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann Hilfe anzeigen lassen		
Voraussetzungen	Script ist gestartet		
Testschritte	1. Get-HWInfoFromILO /?		
Erwartetes Ergebnis	Die Hilfeseite wird angezeigt: NAME Get-HWInfoFromILO SYNOPSIS [...] SYNTAX [...]		

Testfall:	**TF-02**	**Anforderung:**	**ANF-02**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann Hilfe anzeigen lassen		
Voraussetzungen	Script ist gestartet		
Testschritte	1. Get-HWInfoFromILO -configPath 0		
Erwartetes Ergebnis	Die Hilfeseite über Parameter wird angezeigt und auf den Fehler hingewiesen, dass der Pfad nicht richtig ist.		

Testfall:	**TF-03**	**Anforderung:**	**ANF-03, ANF-05**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann Hilfe anzeigen lassen		
Voraussetzungen	Script ist gestartet & Konfiguration vorhanden, Loglevel ist keine Zahl		
Testschritte	1. Get-HWInfoFromILO -configPath C:\PathToConfig		
Erwartetes Ergebnis	Die Hilfeseite über Parameter wird angezeigt und auf den Fehler hingewiesen, dass der Loglevel keine Zahl sein kann.		

Testfall:	**TF-04**	**Anforderung:**	**ANF-06**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann eine Leere Konfiguration generieren		
Voraussetzungen	Script ist gestartet, keine Konfigurationsdatei hinterlegt.		
Testschritte	1. Get-HWInfoFromILO 2. Aus Optionen “1- Generate empty config» wählen 3. Pfad C:\Users\wernle_y\Downloads eingeben		
Erwartetes Ergebnis	Leere Konfigurationsdatei «config.json» wird im spezifizierten Pfad erstellt.		

Testfall:	**TF-05**	**Anforderung:**	**ANF-07**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann eine Konfiguration generieren, welche Beispieldaten beinhaltet		
Voraussetzungen	Script ist gestartet, keine Konfigurationsdatei hinterlegt.		
Testschritte	1. Get-HWInfoFromILO 2. Aus Optionen “2 - Generate dummy-config» wählen		
Erwartetes Ergebnis	Konfigurationsdatei «config.json» wird erstellt und enthält Beispiel-daten.		

Testfall:	**TF-06**	**Anforderung:**	**Keine**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann eine Konfiguration neu hinterlegen		
Voraussetzungen	Script ist gestartet, keine Konfigurationsdatei hinterlegt.		
Testschritte	1. Get-HWInfoFromILO 2. Aus Optionen “3 – Add Path to Configuration» wählen 3. Set-ConfigPath -Path “C:\Users\wernle_y\Downloads\con-fig.json 4. Get-Config		
Erwartetes Ergebnis	Es wird die Konfiguration vom spezifizierten Pfad zurückgegeben.		

Testfall:	**TF-07**	**Anforderung:**	**ANF-02**
Testart	Funktionaler Test		
Beschrieb	Benutzer kann eine Konfiguration neu Hinterlegen		
Voraussetzungen	Script ist gestartet, keine Konfigurationsdatei hinterlegt.		
Testschritte	1. Get-HWInfoFromILO 2. Aus Optionen “3 – Add Path to Existing Config” 3. “A:\IPA\IPA” eingeben (Pfad, der nicht existiert)		
Erwartetes Ergebnis	Fehler wird geloggt: Pfad konnte nicht gefunden werden		

Testfall:	**TF-08**	**Anforderung:**	**ANF-10, ANF-16**
Testart	Funktionaler Test		
Beschrieb	Führe eine InventoryAbfrage durch.		
Voraussetzungen	Script ist gestartet, eine richtige Konfigurationsdatei ist hinterlegt, Inventory ist erreichbar.		
Testschritte	1. Option in Konfig «SearchStringInventory» und «remoteMgmntField» sind auf «gfa-sioc-cs-de» und «HostnameMngt» gesetzt. «IgnoreMACAddress» und «IgnoreSerialNumber» sind nicht gesetzt. 2. Get-HWInfoFromILO		
Erwartetes Ergebnis	Es werden die Servernamen aus Inventory abgefragt, und in ein .json-File zwischengespeichert. Danach wird anhand der Hostnamen ein Pingtest durchgeführt --> als Resultat werden ein report.json und dreu Csv (generell.csv, mac.csv und SerialNumber.csv generiert. Das Resultat enthählt MAC-Adressen und Seriennummern in CSV-Dateien und einer JSON-Datei		

Testfall:	**TF-09**	**Anforderung:**	**ANF-10, ANF-24**
Testart	Funktionaler Test		
Beschrieb	Teste ob wenn Inventory nicht erreichbar ist der richtige Fehler angezeigt wird.		
Voraussetzungen	Script ist gestartet, eine richtige Konfigurationsdatei ist hinterlegt, Inventory nicht erreichbar.		
Testschritte	1. Option in Konfig «SearchStringInventory» und «remoteMgmntField» sind auf «gfa-sioc-cs-de» und «HostnameMngt» gesetzt. «IgnoreMACAddress» und «IgnoreSerialNumber» sind nicht gesetzt. 2. Get-HWInfoFromILO		
Erwartetes Ergebnis	Das Script startet komplett normal, gibt aber dann den Fehler zurück, dass Inventory die Verbindung abgelehnt hatte.		

Testfall:	**TF-10**	**Anforderung:**	**ANF-11, ANF-12**
Testart	Funktionaler Test		
Beschrieb	Teste ServerPath wenn in Konfiguration so spezifiziert.		
Voraussetzungen	Script ist gestartet, eine richtige Konfigurationsdatei ist hinterlegt, in der Konfiguration ist ein Feld «serverPath» mit einem Pfad zu einem Array an Servernamen. Inventory ist ausgeschaltet.		
Testschritte	1. Option in Konfig «doNotSearchInventor»ist auf true gesetzt. «IgnoreMACAddress» und «IgnoreSerialNumber» sind beide gesetzt. serverPath ist gesetzt auf einen Pfad, in dem sich ein Server.json mit der Value «[rmgfa-sioc-cs-dev]» befindet 2. Get-HWInfoFromILO		
Erwartetes Ergebnis	Die Server werden aus der Datei abgefragt, ein Pingtest durchgeführt und eine ILO-Abfrage durchgeführt. Es wird am Ende ein Report.json erstellt.		

Testfall:	**TF-11**	**Anforderung:**	**ANF-11, ANF-08**
Testart	Funktionaler Test		
Beschrieb	Führe Anfrage rein per Parameter durch, mit einem Array an Servern als Parametern		
Voraussetzungen	Script ist gestartet, keine Konfigurationsdatei ist hinterlegt.		
Testschritte	1. Get-HWInfoFromILO -server @("rmgfa-sioc-cs-dev", "rmgfa-sioc-cs-de4", "rmgfa-sioc-cs-de3") -Password (ConvertTo-SecureString -String "test!1234" -AsPlainText) -Username "Yannik" -DeactivateCertificateValidationILO;		
Erwartetes Ergebnis	Die Server werden aus dem Array in eine Datei zwischengespeichert, ein Pingtest durchgeführt und eine ILO-Abfrage durchgeführt. Es wird am Ende ein Report.json erstellt.		

Testfall:	**TF-12**	**Anforderung:**	**ANF-08**
Testart	Funktionaler Test		
Beschrieb	Führe eine Abfrage rein per Parameter durch, mit einer Konfiguration als Parameter.		
Voraussetzungen	Script ist gestartet, es exisitiert eine Konfiguration unter «C:\Users\wernle_y\Download\config.json», es wurde aber sonst keine Hinterlegt.		
Testschritte	1. Get-HWInfoFromILO -configPath “C:\Users\wernle_y\Download\config.json”		
Erwartetes Ergebnis	Die Abfrage findet komplett statt, verwendet die Konfiguration und die Dateien werden richtig generiert.		

Testfall:	**TF-13**	**Anforderung:**	**ANF-08, ANF-11, ANF-12**
Testart	Funktionaler Test		
Beschrieb	Führe eine Abfrage rein per Parameter aus, mit einer Datei wo die Server abgelegt sind.		
Voraussetzungen	Script ist gestartet, es exisitiert ein Server.json-Datei unter «C:\Users\wernle_y\Download\ Server.json» mit dem Wert «rmdl20test».		
Testschritte	1. Get-HWInfoFromILO -serverPath “C:\Users\wernle_y\Download\server.json”		
Erwartetes Ergebnis	Die Abfrage findet komplett statt auf Grundlage der Server.json statt und die Dateien werden richtig generiert.		

Testfall:	**TF-14**	**Anforderung:**	**ANF-08, ANF-10**
Testart	Funktionaler Test		
Beschrieb	Führe eine Abfrage rein per Parameter und per Inventory durch.		
Voraussetzungen	Script ist gestartet, Inventory ist erreichbar.		
Testschritte	1. Get-HWInfoFromILO -SearchStringInventory “gfa-sioc-cs-de”		
Erwartetes Ergebnis	Die Abfrage durchsucht Inventory aufgrund des Suchstrings, speichert die beiden Server «rmgfa-sioc-cs-dev» und «rmgfa-sioc-cs-de4» in einer Datei und führt aufgrund von ihr die Abfrage aus. Es werden alle Dateien generiert.		

Testfall:	**TF-15**	**Anforderung:**	**ANF-04**
Testart	Funktionaler Test		
Beschrieb	Anzeige einer Warnung bei abweichender Version		
Voraussetzungen	Script ist gestartet, die Bibliothek HPEiLOCmdlets ist in einer anderen Version als 4.4.0.0 installiert		
Testschritte	1. Get-HWInfoFromILO -SearchStringInventory “rmgfa-sioc-cs-de”		
Erwartetes Ergebnis	Warnung: Die Verwendete Bibliotheksversion ist nicht die empfohlene Version		

Testfall:	**TF-16**	**Anforderung:**	**ANF-09**
Testart	Funktionaler Test		
Beschrieb	Logfunktion		
Voraussetzungen	Script ist gestartet, eine Konfigurationsdatei ist konfiguriert mit Loglevel 1, ein LogPfad ist gesetzt & die Aktivierung ist auf true gesetzt		
Testschritte	1. Get-HWInfoFromILO		
Erwartetes Ergebnis	Es werden die gröberen Schritte des Programms am spezifizierten Ort in der Log.txt abgespeichert.		

Testfall:	**TF-17**	**Anforderung:**	**ANF-18, ANF-19, ANF-20, ANF-21, ANF-22, ANF-23**
Testart	Funktionaler Test		
Beschrieb	Überprüfung der Speicherung in CSV		
Voraussetzungen	Script ist gestartet, die Konfigurationsdatei enthält ignore-MACAddresses = true und einen ReportPath		
Testschritte	1. Get-HWInfoFromILO		
Erwartetes Ergebnis	Es werden 2 CSV-Dateien generiert (generell.csv & SerialNumber.csv). Beim Verbindungsaufbau per ILO wird der Stand angezeigt und haben einen Zeitstempel		

Testfall:	**TF-18**	**Anforderung:**	**ANF-17**
Testart	Funktionaler Test		
Beschrieb	Überprüfung ob Pingtest ausschaltbar ist		
Voraussetzungen	Script ist gestartet, die Konfigurationsdatei korrekt konfiguriert und enthält «deactivatePingtest=true».		
Testschritte	1. Get-HWInfoFromILO		
Erwartetes Ergebnis	Es wird kein Pingtest durchgeführt, was durch den fehlenden Output in der Konsole ersichtlich ist.		

2.3.3 Verwendete Technologien

Bei der Implementation des Scripts werden die folgenden Technologien, Bibliotheken und Programme verwendet:

- PowerShell v.7.5.0
- HPEiLOCmdlets v.4.4.0.0
- Visual Studio Code v.1.97.0
- VS-Code PowerShell Extension (2025.0.0)
- Git-Extensions v.5.2.1

In der Dokumentation und den Darstellungen darin kamen die folgenden Programme zur Anwendung:

- Microsoft Onedrive.com (Backup Dokumentation)
- Draw.io (Darstellungen und Diagramme)
- Microsoft Word (Dokumentation)
- Microsoft Excel (Zeitplan und Entscheidungsmatrix)
- PAP-Designer

2.3.4 Konfigurationsdatei(en)

Für die Konfigurationsdatei macht es Sinn, sie in verschiedene kleinere Konfigurationsdateien aufzuteilen. Dies ist vor allem beim Login wichtig, denn wenn man eine Konfigurationsdatei verschickt, will man nicht unbedingt, dass das eigene Passwort mitsamt Benutzername verschickt wird. Es besteht immer das Risiko, dass es abgefangen wird und ein gestohlenes Passwort kann viel Schaden anrichten.
Als Resultat davon soll das Login in einer separaten Konfiguration gespeichert werden – so kann man die *normale* Konfiguration ohne Bedenken weitersenden.

Ein weiterer Teil, welcher in ein separates File gehört, ist die Liste an Servern, die abgefragt werden soll. Wenn Inventory erreichbar ist, wird diese Liste automatisch aufgefüllt, aber wenn es nicht erreichbar ist, dann muss der Nutzer diese selbst ausfüllen. So kann man die in der Aufgabenstellung geforderte Zwischenspeicherung einfach machen und es trotzdem sinnvoll und schnell änderbar machen.

Als Dateiformat der Konfiguration bietet sich JSON sehr gut an – es ist weit verbreitet und unterstützt. Ausserdem ist es gut lesbar und einfach änderbar.

Haupt-Konfigration («config.json»)

Bezeichnung	Wert
searchForFilesAt	C:\Path\To\Somewhere
configPath	C:\Path\To\Somewhere
loginConfigPath	C:\Path\To\Somewhere
reportPath	C:\Path\To\Somwhere
logPath	C:\Path\To\Somwhere
serverPath	C:\Path\To\Somwhere
logLevel	0

loggingActivated	True
searchStringInventory	sf-sioc-cs
doNotSearchInventory	false
remoteMgmntField	Hostname Mgnt
deactivateCertificateValidationILO	false

Die Bezeichnungen sind weitestgehend selbsterklärend – sie sind jeweils der Pfad, wo entweder eine Datei abgelegt ist oder wo Dateien abgelegt werden sollen.

Besonders zu erwähnen sind die Felder «searchStringInventory» und «RemoteMgmntField». Ersteres gibt an, mit welchem String gesucht werden soll, und letzteres ist das Feld in Inventory, welches schlussendlich zurückgegeben werden soll.

Als weiteres wichtiges Feld ist «serverPath» – dort werden die server von Inventory zwischengespeichert, resp. dort muss man die Hostnamen der Server ablegen, um sie via ILO abzufragen.

Falls das der Benutzer wünscht, kann er mittels «deactivateCertificateValidationILO» die Validation der Zertifikate deaktivieren, sodass er kein gültiges Zertifikat braucht, um mit iLO zu kommunizieren.

Login-Konfiguration («login.json»)

Bezeichnung	Wert
Username	SomeUsername
Password	SomePassword

Server-Konfiguration («servers.json»)
Die Serverkonfiguration unterscheidet sich von den anderen beiden Konfigurationen darin, dass es eigentlich nur ein Array an Hostnamen ist und keine weiteren Eigenschaften enthält.

```
[
      «server-hostname-1»,
      «server-hostname-2»,
      «server-hostname-3»,
      «server-hostname-4»,
      «server-hostname-5»,
]
```

Codesnippet 1: Inhalt der Server-Konfiguration

2.3.5 Parameter

PowerShell ermöglicht viele verschiedene Arten, Parameter zu verwenden. Die Parameter sind essenziell eigentlich die der Konfigurationsdateien.

Im Prinzip geht es darum herauszufinden, welche Parameter zwingend nötig sind, um das Script per Kommandozeile und ohne Konfiguration starten zu können: Dies sind eigentlich nur der Benutzername und Passwort für ILO. Die anderen Parameter sind alles solche, die für das Script genutzt werden können, aber nicht immer notwendig sind.

Es gibt insgesamt vier Fälle, um das Script per Kommadozeile zu starten:
1. via Verbindung auf Inventory (d.h. Angabe von «searchStringInventory», «Username»**,** **«**Password»)
2. via direkter Angabe einer Liste an Servern (d.h. Angabe eines Arrays zur Suche, «Username», «Password»)
3. via Angabe einer Liste an Servern als Datei (d.h. «serverPath», «Username», «Password»)
4. via Angabe einer Konfiguration (d.h. «configPath», «Username», «Password»)

Aus diesen Überlegungen lässt sich diese Parameterkonfiguration ableiten:

```
[CmdletBinding(PositionalBinding = $False)]
param(
    [Parameter(Mandatory = $true,
        ParameterSetName = "Config")]
    [string]
    $ConfigPath,

    [Parameter()]
    [string]
    $LoginConfigPath,

    [Parameter()]
    [string]
    $ReportPath,

    [Parameter()]
    [string]
    $LogPath,

    [Parameter(Mandatory = $true,
        ParameterSetName = "ServerPath")]
    [string]
    $ServerPath,

    [Parameter(Mandatory = $true,
        ParameterSetName = "ServerArray")]
    [array]
    $server,

    [Parameter()]
    [int]
    $LogLevel,

    [Parameter()]
    [switch]
    $LoggingActivated,

    [Parameter(Mandatory = $true,
        ParameterSetName = "Inventory")]
    [string]
    $SearchStringInventory,

    [Parameter()]
    [switch]
    $DoNotSearchInventory,

    [Parameter()]
    [string]
    $RemoteMgmntField,

    [Parameter()]
    [switch]
    $DeactivateCertificateValidationILO,

    [Parameter(Mandatory = $true,
        ParameterSetName = "Config")]
```

```
    [Parameter(Mandatory = $true,
        ParameterSetName = "ServerPath")]
    [string]
    $Username,

    [Parameter(Mandatory = $true,
        ParameterSetName = "Config")]
    [Parameter(Mandatory = $true,
        ParameterSetName = "ServerPath")]
    [Parameter(Mandatory = $true,
        ParameterSetName = "ServerArray")]
    [Parameter(Mandatory = $true,
        ParameterSetName = "Inventory")]
    [string]
    $Password
)
```

Codesnippet 2: Geplante Parameter

Dabei wird anhand der oben beschriebenen vier Fälle je ein Parameterset erstellt – das bedeutet, es kann nur jeweils ein einziges davon gleichzeitig verwendet werden. Man kann also nicht z.B. ein ServerArray angeben & ein Server-Path, das wird damit verhindert.

Die Datentypen sind ebenfalls so gut als möglich den Datentypen angepasst, welche später vermutlich verwendet werden.

Die Nutzung des **Positionalbinding**-Arguments ganz am Anfang der Parameterliste führt dazu, dass die Parameter nicht positionell gesetzt werden können, sondern nur durch genaue Angabe des Namens. Dadurch wird der Nutzer dazu gebracht, darüber nachzudenken, was er da überhaupt eingibt und zusätzlich ist es im Nachhinein auch besser nachvollziehbar, welcher Wert wofür verwendet wurde.

2.3.6 Programmablaufplan detailliert

Aus dem simplen Programmablaufplan lässt sich zwar ableiten, wie die grobe Funktionalität des Scriptes am Ende aussehen soll, aber als Template für die Implementation ist es nicht geeignet – es ist zu wenig detailliert und zu verallgemeinert.

Um es als grobes strukturelles Template verwenden zu können braucht es mehr Details. Anhand des bisherigen Templates und der Anforderungsanalyse ergibt sich das folgende PAP:

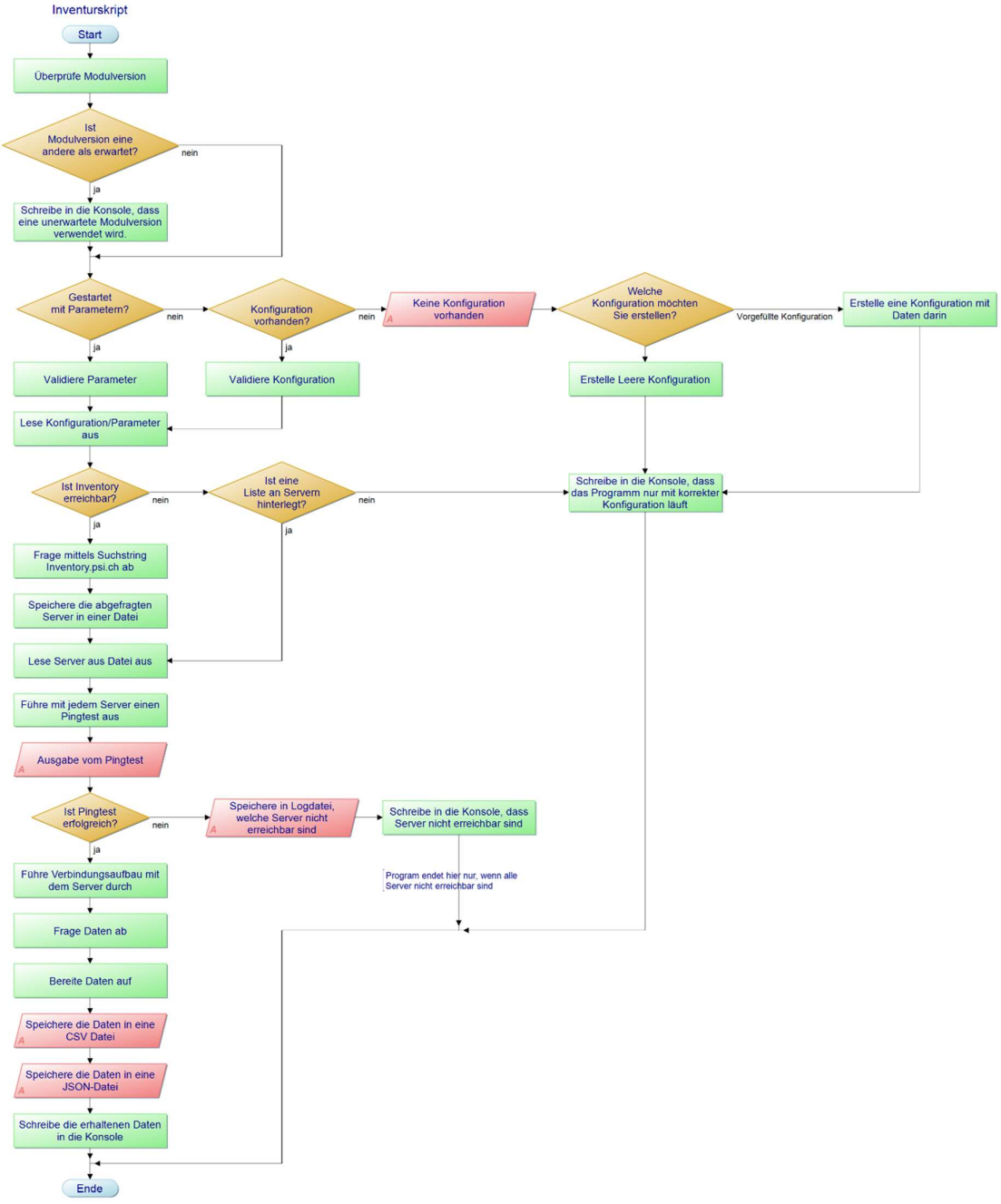


Abbildung 11: Programmablaufplan Script detailliert

2.4 Entscheiden

Aus der Aufgabenstellung gibt es eigentlich nur eine vorgeschriebene Entscheidung: Welche Parameter sind nötig zum Starten des Scripts. Mittels der Parametersets in PowerShell lassen sich mehrere verschiedene Optionen gleichzeitig implementieren – so lassen sich verschiedene Bedürfnisse gleichzeitig abdecken, ohne dass man auf einen einzelnen beschränkt wird.

Es gibt jedoch noch eine andere Entscheidung in Bezug auf die Programmierstruktur – PowerShell lässt die Implementierung von Code sowohl als Modul als auch als einzelnes Script (oder mehrere Dateien) zu, was natürlich je nach Option einige Dinge stark vereinfacht oder komplizierter macht. Die Aufgabenstellung stellt keine besonderen Anforderungen an die Struktur des Scriptes und der Fachvorgesetzte war damit einverstanden, dass ich das auch so implementieren kann. Anhand der Entscheidungsmatrix soll nun entschieden werden, welche der beiden Optionen die bessere ist.

Entscheidungsmatrix Programmstruktur

		Lösungskonzepte			
Kriterien		Script		Modul	
Anforderungen	**Gewichtung**	**Max = 5**		**Max = 5**	
			Nutzwert		**Nutzwert**
Lesbarkeit in grossen Projekten	**3**	3	**9**	4	**12**
Einfachheit d. Entwicklung	**2**	5	**10**	3	**6**
Unabhängigkeit von Internet	**2**	4	**8**	3	**6**
Einfachheit d. Deployments	**1**	3	**3**	5	**5**
Anpassbarkeit	**1**	3	**3**	5	**5**
Wiederverwendbarkeit Code	**2**	2	**4**	5	**10**
TOTAL			**37**		**44**

Tabelle 18: Entscheidungsmatrix Programmierstruktur

Fazit

Wie man anhand der Entscheidungsmatrix unschwer erkennen kann, ist die Entscheidung knapp ausgefallen und auf dem Modul gelandet. Das ist vor allem der Wiederverwendbarkeit des Codes sowie der Lesbarkeit geschuldet.

Als Script ist gemeint, mehrere Dateien zu verwenden, aber diese nicht durch ein Modul zentral zu bündeln. Es ist zwar einfacher in der Entwicklung, aber dafür geht die Modularität und die Wiederverwendbarkeit verloren – da dieses Script später noch weiterentwickelt werden soll, ist diese aber stark von Vorteil. Ansonsten ist das Modul in den meisten Bereichen nur leicht besser als das Script. Ein Nachteil des Moduls ist, dass der Kandidat noch nie selbst ein komplettes Modul entwickelt hat, sondern nur zum Testen.
Das Modul bietet ausserdem als entscheidenden Vorteil auch ein vereinfachtes Deployment mit sich – es kann mit Import-Module von einem zentralen Repository installiert werden, ausserdem können Funktionen gezielt ein- oder ausgeblendet werden. Ein Nachteil ist zugleich aber, das für die Ausführung des Programmes nun nicht über eine einzelne Zeile gestartet werden kann, sondern nun mit 2 Zeilen. Dies ist aber nicht so dramatisch und es kann darüber hinweggesehen werden.

2.5 Realisieren

2.5.1 Grundstruktur

Die Grundstruktur des Programms ist nach den Standards von PowerShell-Modulen aufgebaut:

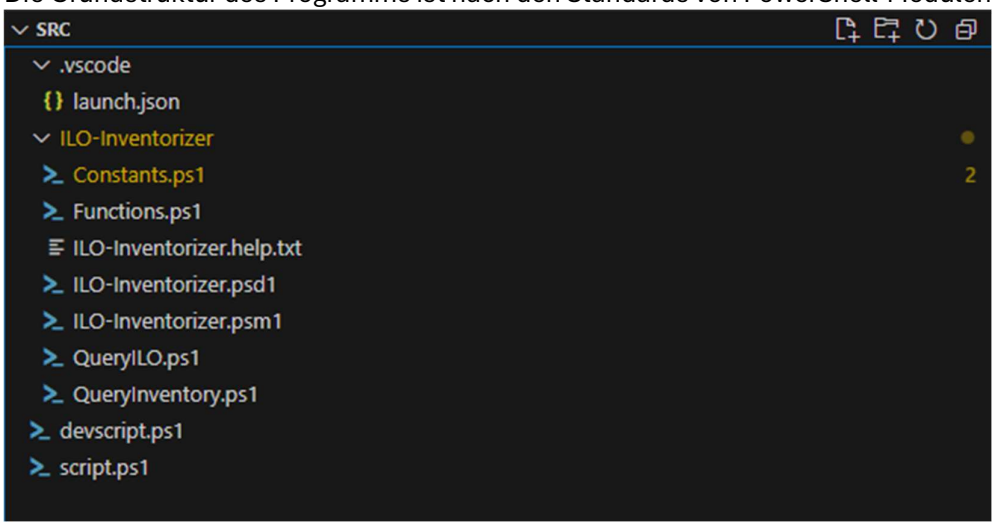


Abbildung 12: Ordnerstruktur

Der Ordner **ILO-Inventorizer** wird dabei als Wrapper für das Modul verwendet, mit der gleichnamigen Moduldatei (**.psm1**) darin. Ausserdem sind dort weitere Scripts angelegt, die spezifische Funktionen beinhalten, um so den Überblick besser zu behalten.
Die **ILO-Inventorizer.help.txt**-Datei ist dazu da, die Modulhilfe abzufangen (**Get-Help ILO-Inventorizer**) und leitet in ihrem enthaltenen Text in auf die eigentlichen Funktionen und Hauptfunktionen weiter.

Zur vereinfachten Entwicklung gibt es das **launch.json**, welches die Konfiguration fürs Debuggen im VS Code enthält – diese ruft eigentlich nur das **script.ps1** auf, welches jeweils das Modul neu lädt und die Start-Funktion aufruft. Dies ist dazu gedacht, das Script einfacher debuggen zu können:

```
Remove-Module ILO-Inventorizer;
Import-Module .\ILO-Inventorizer\ILO-Inventorizer.psm1;
Get-HWInfoFromILO
```

Codesnippet 3: Inhalt des Startscripts

Es soll am Ende des Scriptes nur eine kleine Anzahl an Funktionen nach Aussen erreichbar sein – wie oben erkennbar ist eine davon die **GetHWInfoFromILO**. Diese soll den ganzen Dialog und die eigentliche Abfrage triggern.

Da für die Ausführung des Programms eine externe Bibliothek nötig ist, wird sie ebenfalls hier überprüft – falls sie eine andere als erwartet ist, dann wird darauf hingewiesen und davor gewarnt.

Die Grundstruktur für die Hilfe ist ebenfalls bereits implementiert: Sie wird durch zwei Parameter gelöst, wobei der eine davon help ist und die Parameter **--h**, **--help** und **/?** Abfängt. Da PowerShell zur Kennzeichnung von Parametern **'-'** verwendet, muss **-h** als Parameter implementiert werden und kann nicht wie die anderen direkt als String abgefangen werden.

Das ist sehr einfach durch den Parametertyp [switch] gelöst – es wird im Hintergrund einfach etwas getrickst, um trotzdem die Hilfe-Funktion aufzurufen.

```
## Check if Help must be displayed
if (($h -eq $true) -or ((Show-Help $help) -and ($help.Length -gt 0)) ) {
    Get-Help Get-HWInfoFromILO -Full;
}
```

Codesnippet 4: Get-Help Funktionalität

Die aufgerufene Show-Help Funktion macht im Grunde auch nichts anderes, als Get-Help Get-HWInfoFromILO -Full aufzurufen. Das bedeutet auch, dass die native Art, Hilfe anzufordern in PowerShell selbstverständlich auch läuft.

2.5.2 Modulversion

Die Modulversion, auf der alles getestet wurde, ist die Version 4.4.0.0 des HPEiLOCmdlets. Dies ergibt sich aus der Erfahrung der Test-IPA.

Falls eine andere Version verwendet wird, soll das aber kein Problem sein – in Absprache mit dem VF wird bei abweichenden Versionen lediglich der User darauf hingewiesen und ihm gesagt, dass es zu Fehlern aufgrund der Versionsdifferenzen geben kann.

```
Import-Module HPEiLOCmdlets;
## Check for recommended ModuleVersion
$moduleVersion = (Get-Module -Name HPEiLOCmdlets).Version.ToString()
if ($recommendedVersion -ne ($moduleVersion)) {
            Write-Warning "The installed Module HPEiLOCmdlets doesnt use the recommended
Version '$recommendedVersion', but '$moduleVersion' - some features may not work cor-
rectly."
}
```

Codesnippet 5: Überprüfung Modulversion

2.5.3 Generierung der Konfigurationsdatei

Die Generierung der Konfigurationsdateien lässt sich grob auf zwei verschiedene Aspekte aufteilen. Das ist einerseits die Eingabe auf Seiten des Users, der sich entscheiden muss, was für eine Konfiguration er denn überhaupt haben will, was man als «Frontend» bezeichnen könnte.

Die Umsetzung des «Frontends» ist im Endeffekt nichts anderes als verschachtelte Switch-Case-Statements, welche die 4 verschiedenen Möglichkeiten der Generierung abdecken und generell diesem Muster an Nutzereingaben (Read-Host) und switch-Statements folgen:

```
$pathToSaveAt = Read-Host -Prompt "Where do you want to save the config at?";
$withInventory = Read-Host -Prompt "Do you want to:`nRead From Inventory
[y/N]?"
switch ($withInventory) {
    "y" {
        New-Config -Path $pathToSaveAt -NotEmpty;
        break;
        }
    "N" {
        New-Config -Path $pathToSaveAt -WithoutInventory -NotEmpty;
        break;
        }
}
```

Codesnippet 6: "Frontend" Generierung der Konfigurationsdateien

Im Hintergrund wird anhand der Eingaben dann ein PowerShell-Objekt erstellt und mit den entsprechenden Werten (Dummy-Werte oder leere Werte) gefüllt und anschliessend mittels ConvertTo-Json & Out-File zuerst in ein JSON-Format konvertiert und danach in einer **config.json**-Datei abgespeichert.

Der Pfad zur Konfigurationsdatei wird schlussendlich in einer Umgebungsvariable namens $ENV:HPEILOCONFIG gespeichert. Über sie läuft die eigentliche Abfrage der Konfigurationsparameter ab.

Wenn man sich nicht sicher ist, welche Datei jetzt gerade als Konfigurationsdatei festgelegt ist, gibt es zwei weitere Funktionen: Mit Get-ConfigPath lässt sich die momentan ausgewählte Konfigurationsdatei anzeigen oder mit Set-ConfigPath unkompliziert ändern.

2.5.4 Start via Parameter

Der Start via Parameter unterscheidet sich wenig von der Nutzung mit einer Konfigurationsdatei – es wird im Hintergrund nämlich aus den Parametern eine **temporäre** Konfigurationsdatei erstellt, welche gleich strukturiert ist wie eine normale. Der Unterschied liegt vor allem darin, dass diese bei jedem Start mit Parametern wieder überschrieben wird. Diese Implementation wurde so implementiert, um im Hintergrund überall die gleiche Struktur zu wahren und die späteren Abfragen von Inventory und den ILO-Systemen zu vereinfachen.

PowerShell bietet mit ParameterSets eine gute Möglichkeit verschiedene Arten, wie das Script gestartet wurde, zu unterscheiden.
Es lässt sich das Komplizierteste über die Parameter lösen – es muss definiert werden, welche Parameter unabdingbar sind und welche für alle Arten zugänglich sein müssen. Das ist teils zwar umständlich (wie bspw. beim untenstehenden Benutzernamen), aber verkürzt den restlichen Teil später erheblich.

```
        [Parameter(Mandatory = $true,
            ParameterSetName = "ServerPath")]
        [Parameter(Mandatory = $true,
            ParameterSetName = "ServerArray")]
        [Parameter(Mandatory = $true,
            ParameterSetName = "Inventory")]
        [Parameter(
            ParameterSetName = "Config")]
        [Parameter(
            ParameterSetName = "None")]
        [string]
        $Username,
```

Codesnippet 7: ParameterSet am Benutzer

Wie oben schon zu sehen ist, gibt es im Grunde fünf verschiedene Arten. Diese sind deckungsgleich mit der Planung. Einzig das ParameterSet «None» ist neu – dieses ist nötig, dass das Script auch ohne Parameter gestartet werden kann.
Aus diesen Parametersets ergibt sich auch die Syntax:

```
Get-HWInfoFromILO -configPath <string> [Other Parameters]
Get-HWInfofromILO -ServerPath <string> -Username <string> -Password <string> [Other Parameters]
Get-HWInfofromILO -server <array> -Username <string> -Password <string> [Other Parameters]
Get-HWInfofromILO -SearchStringInventory <string> -Username <string> -Password <string> [Other Parameters]
```

Codesnippet 8: Syntax Get-HWInfofromILO

Das Handling der Parametersets erfolgt durch ein einfaches Switch-Case. Erwähnenswert ist auch die Funktion, welche die Änderungen speichert, die zusätzlich mitgegeben werden (z.B. mit «configPath» gestartet, aber mit der Änderung, dass das `logLevel = 0` ist. Im Hintergrund überprüft die Update-Funktion dann vorhandene Änderungen und speichert diese ab.

```
Update-Config -configPath $configPath -LoginConfigPath $LoginConfigPath -ReportPath $ReportPath -LogPath $LogPath
```

Codesnippet 9: Beispiel Update-Config

2.5.5 Logfunktion

Loglevels

Im Laufe der Entwicklung wurden die folgenden Loglevels definiert, die von «keine Logs» bis zu fast zeilenweise alles beinhalten.

Level	Bezeichnung
0	Keine Logs
1	Keine Logs, ausser bei Fehlern
2	Logs nur bei Start, Ende des Scripts und bei Fehlern
3	Logs nur bei den wichtigsten Schritten - Auslesen der Konfiguration - Pingtest - Abfrage von Inventory - Abfrage von ILO - Speichern in die Dateien - Ende des Scripts - Fehler
4	Genauere Logs bei den wichtigsten Schritten - Auslesen der Konfiguration - Pingtest o Ergebnis der einzelnen Server - Abfrage von Inventory o Ergebnis der einzelnen Server - Abfrage von ILO o Abschluss der einzelnen Server o Abschluss der Abfrage von ILO - Speicherfunktion o In JSON o In CSV - Ende des Scripts - Fehler
5	Genaue Logs der wichtigsten Schritte & Logs bei Beginn und Ende einer aufgerufenen Funktion
6	Genaue Logs der wichtigsten Schritte & genaue Logs der Funktionen.

Implementierung

Die Logfunktion ist zentral geregelt – ausserdem ist sie so implementiert, dass sie keine anderen nichtstandard-Funktionen aufruft – so lässt sie sich überall ohne Probleme einsetzen, ohne dass es zu Rekursiven Funktionen kommt.

Die Funktion erstellt jeweils pro Tag ein einzelnes Logfile, in welchem alle Vorkommnisse des Tages erfasst sind. Die Speicherung erfolgt jeweils mit Zeitstempel und die einzelnen Ausführungen werden durch eine Linie getrennt.

```
2025.03.10 14:30:49     ------------------------------------
ILO-Inventorizer has been started.
2025.03.10 14:30:49     Configure new Configuration
```

```
2025.03.10 14:30:49     Import Configuration
2025.03.10 14:30:49     Start Updating Configuraton File
```

Codesnippet 10: Ausschnitt Logdatei

2.5.6 Pingtest

Der Pingtest besteht aus zwei Teilen: Einem nslookup und einem Test-Connection – die Teile geben unterschiedliche Informationen zurück, die für den Nutzer beim Problemlösen sehr nützlich sein können:

- Nslookup stellt eine Anfrage an den DNS-Server, wodurch sich sagen lässt, ob ein abgefragter Server oder Computer überhaupt existiert
- Test-Connection wird dann dazu verwendet, um zu prüfen, ob man den anderen Server auch erreichen kann.

Das heisst für den User, dass wenn schon beim nslookup keine Antwort kommt, dann ist der Server im Netzwerk nicht erreichbar oder nicht vorhanden. Gibt es hingegen beim Test-Connection keine Antwort (trotz Auftauchen im lookup), bedeutet das lediglich, dass man keinen Zugriff auf das Teilnetz hat.

Der Pingtest wird per Loop mit allen eingegebenen Systemen durchgeführt, wobei die Systeme abgefangen werden, welche nicht erreichbar sind und somit auch eine ILO-Abfrage keinen Sinn machen würde.

```
        [Array]$reachable;
        foreach($srv in $serverJSON){
            if(Invoke-PingTest $srv){
                $reachable += $srv;
            }
        }
```

Codesnippet 11: Aufruf Pingtest

2.5.7 Abfrage von Inventory

Die Abfrage von Inventory wird dazu benötigt, anhand eines Suchstrings aus der Datenbank die ILO-Hostnamen zu finden. Hierzu wird zuerst überprüft, ob gemäss Konfiguration Inventory überhaupt ausgelesen werden soll.

Wenn dies Fall ist, wird zuerst mittels der bereits implementierten Funktion ein Pingtest an Inventory.psi.ch durchgeführt, um die Erreichbarkeit zu verifizieren. Anschliessend wird die eigentliche Abfrage durchgeführt, in der neben dem Hostnamen für ILO auch die Bezeichnung im Inventory abgefragt wird, die später für die CSV-Dateien wichtig sind.

Als letztes werden schliesslich der Hostname für die ILO-Abfrage zur Nachverfolgung im server.json gespeichert, von dem sie später wieder ausgelesen werden.

Die Abfrage von Inventory verwendet die bereitgestellte JSON-API unter IV4 - JSON API Introspection, genauer gesagt den Endpoint «FindObjects», der für die generelle Suche verwendet werden kann. Dieser ist am besten geeignet, da er frei konfigurierbar ist was die durchsuchten Felder und Operatoren angeht.

```
$uri = "https://inventory.psi.ch/DataAccess.asmx/FindObjects";
$headers = @{"Content-Type" = "application/json; charset= utf-8" };
$body = @{
       "search" = @{
```

```
            "query"   = @(@{
                   "Field"    = "ANY"
                   "Operator" = "Contains"
                   "Value"    = $searchStringInventory
                   })
            "columns" = @(
                  "Label",
                  "Hostname",
                  $remoteMgmntField,
                  "Serial",
                  "Part Type",
                  "Facility",
                  "MAC 1",
                  "MAC 2",
                  "MAC 3",
                  "MAC 4",
                  "Mgnt MAC",
                  "HW Status",
                  "OS"
                  )
               }
        } | ConvertTo-Json -Depth 4

$resp = Invoke-RestMethod -Uri $uri -Method Post -Headers $headers -Body $body -HttpVersion
3.0
```

Codesnippet 12: Abfrage Inventory

Es werden neben den Hostnamen weitere Informationen abgespeichert, wie die MAC-Adressen oder die Seriennummer, die nicht direkt für die ILO-Abfrage nützlich sind, aber für den Nutzenden zur Konfiguration der Computer sehr hilfreich sein kann.

Um für die Abspeicherung der CSV-Dateien einfacher an das *Label*-Feld zu kommen, braucht es eine Funktion, welche die Antwort von Inventory vom unschönen verschachtelten Array zu einem schönen Array aus Objekten konvertiert:

```
            $servers = (($resp).d.Rows);
            $serversClean = @();
            foreach ($srv in $servers) {
                $serversClean += [ordered]@{
                    Label         = $srv[0]
                    Hostname      = $srv[1]
                }
            }
```

Codesnippet 13: Konvertierung der Inventory-Antwort

Dieses Array wird dann zu JSON konvertiert und in einer Datei für später und zur Nachverfolgung zwischengespeichert. Aus dieser Datei stammen bspw. das Label, welches später bei der CSV-Speicherung wieder wichtig wird.

2.5.8 Abfrage von ILO

Die Abfrage von ILO ist simpel aufgebaut – sobald mit Connect-HPEILO die Verbindung aufgebaut ist, lässt sich durch Pipeing der jeweilige Server abfragen. Probleme gibt es vor allem zwischen den Versionen von ILO – bei den verwendeten DL380-Servern der 8. Generation beispielsweise gibt es einige Funktionen, die nicht genutzt werden können (z.B. PCI-Devices). Ausserdem sind teilweise die gleichen Funktionen teilweise je nach Version anders strukturiert.

Aus diesem Grund und daraus, dass nur die wichtigsten Dinge abgespeichert werden sollen, ist ein grosser Teil dieses Codes einfach nur dazu nötig, um über alle abgedeckten ILO-Versionen (4-6) die Eigenschaften zu vereinheitlichen und ansatzweise zu standardisieren.

Der Teil, der die Informationen des Memories ausliest, sieht beispielsweise so aus:

```
Log 6 "`tQuerying Memory" -IgnoreLogActive
$memory = $iLOVersion -eq 4 ? ($conn | Get-HPEiLOMemoryInfo).MemoryComponent : ($conn |
Get-HPEiLOMemoryInfo).MemoryDetails.MemoryData;
$memoryDetails = @();
foreach ($me in $memory) {
       $memoryDetails += [ordered]@{
              Location = $iLOVersion -eq 4 ? $me.MemoryLocation.ToString() : $me.DeviceLo-
cator.ToString();
              SizeMB   = $iLOVersion -eq 4 ? $me.MemorySizeMB.ToString() : $me.Capaci-
tyMiB.ToString();
       }
}
```

Codesnippet 14: ILO-Abfrage & Standardisierung von Memory

Die Eigenschaften, welche ausgelesen und standardisiert werden beschränken sich, wenn möglich auf die Seriennummern, MACAdressen und Standortinformationen.

Script-Messages mit Zeitstempel
Eine der Anforderungen ans Programm ist unter anderem, dass die Abfrage des Servers mittels Messages gemacht wird, welche zusätzlich zum Text auch den Zeitstempel beinhalten – das lässt sich durch eine kleine Anpassung der Logfunktion sehr einfach bewerkstelligen:

Die Logfunktion nutzt bereits Zeitstempel und kann über die Konfiguration ausgeschaltet, sowie in die Konsole geloggt werden – beide Anforderungen sind also bereits vorhanden. Deswegen ist nur eine kleine Modifikation notwendig, nämlich ein neuer Parameter für die Logfunktion: [switch]$IgnoreLogActive.

Ist dieser Parameter gesetzt wird im Hintergrund die Konfiguration ignoriert und trotzdem (in die Konsole) geloggt. Dadurch wird es immer angezeigt, auch wenn in der Konfiguration das gesamte Logging ausgeschalten ist.

```
# Log only if activated
if ($logActive -or $IgnoreLogActive) {
                if ($Level -le $logLevel -or $IgnoreLogActive) {
                 # Code
                 }
# Write to Terminal if configured
```

```
                if ($logToConsoleActive -or $IgnoreLogActive) {
                   Write-Host ($saveString);
                }
}
```

Codesnippet 15: Modifizierte Logfunktion

Kompatibilitätsunterschiede

Da die abgefragten Systeme eine Spanne an verschiedenen ILO-Versionen von Version 4 bis Version 6 abdecken, kommt es zwangsläufig zu Funktionen und Features, welche in der einen Version existiert und in einer anderen nicht.

Falls nur die Struktur anders ist, wird wie oben bereits beschrieben die Struktur angepasst, falls eine Funktion gar nicht verfügbar ist, so wird lediglich die Inkompatibilitätsnachricht abgespeichert.

Dies trifft beispielsweise auf die Devices zu, die vor der ILO4 nicht abgefragt werden konnten, weshalb dort die erwähnte Statusinfo angezeigt wird.

```
$devices = ($conn | Get-HPEiLODeviceInventory);
$deviceDetails = @();
if ($iLOVersion -eq 4) { $deviceDetails = $devices.StatusInfo.Message; }else {
      foreach ($dev in $devices.Devices) {
      }
```

Codesnippet 16: DeviceInventory Inkompatibilität

Die Speicherung sieht dann im Objekt so aus:

```
"Devices": "Feature not supported on iLO3, iLO4 and iLO5 (FW Ver:
1.10,1.11,1.15,1.17).",
```

Codesnippet 17: Ausgabe bei Inkompatibilität

2.5.9 Dateispeicherung

Die Report-Dateien werden am Ende in den Pfad hineingespeichert, welcher unter dem Pfad reportPath in der Konfigurationsdatei abgelegt ist.

Dateiformat : Java**S**cript **O**bject **N**otation (JSON)

Die Dateispeicherung in JSON ist die einfachere und simpel – sie umfasst die gefilterte Ausgabe von ILO. Sie unterscheidet sich in der Implementation nur wenig von der Generierung der Konfigurationsdateien, welche auf dem gleichen Weg generiert werden: Es wird zuerst ein PowerShell-Objekt erstellt, welches dann in JSON konvertiert und schlussendlich in einer Datei gespeichert wird.

```
    $report | ConvertTo-Json -Depth 15 | Out-File -FilePath $name -Force;
```

Codesnippet 18: Dateispeicherung JSON

Um zu garantieren, dass die Ausgaben sich in einem Ordner befinden und nicht einer Datei befinden, wurde eine Funktion implementiert, welche die Datei aus dem Pfad entfernt und den Ordner erstellt, falls dieser nicht existiert:

```
Function Register-Directory {
    param(
        [Parameter(Mandatory = $true)]
        [string]
        $Path,

        [Parameter()]
        [switch]
        $ignoreError

    )

    try{
        if ((-not (Test-Path -Path $Path)) -and $ignoreError) {
            New-Item -Path $Path -Force -ItemType Directory;
        }else{
            throw [System.IO.DirectoryNotFoundException] "The directory at '$Path' does not
exist."
        }

        $isDirectory = (Get-Item ($Path)) -is [System.IO.DirectoryInfo];
        if (-not $isDirectory) {
            $Path = $Path | Split-Path -Parent -Resolve;
        }
        return $Path.ToString();
    }catch{
        Write-Error $_
    }
}
```

Codesnippet 19: Generierung des Pfades falls er nicht existiert.

Nach einem Gespräch mit dem Fachvorgesetzten wurde die Funktion so modifiziert, dass sie lediglich bei der Generation der Konfiguration den Pfad erstellt – in allen anderen Fällen muss der Anwender garantieren, dass der Pfad existiert.

Dateiformat: Comma **S**eperated **V**alues (CSV)

Im Vergleich zur Datenspeicherung in JSON ist diejenige mit CSV komplexer, da sie stärker standardisiert ist und sich nicht wie ein Objekt, sondern wie eine Tabelle verhält – dadurch darf es unter anderem keine Verschachtelungen geben.

Die Ausgabe im CSV ist primär für den Nutzenden gedacht und nicht geeignet dafür, sie durch Automatisierungsprozesse weiter zu verwenden. Sie sind auch aus diesem Grund im Vergleich zur JSON-Datei so kurz als möglich gehalten. Aufgrund dessen ist die CSV-Ausgabe in drei verschiedene Dateien strukturiert:

- **Generell**: Diese Datei stellt +- die Standardansicht in Inventory dar – d.h. sie ist sehr kurz

Label	Hostname	Hostname_Mgnt	Serial	MAC_1	MAC_2	MAC_3	MAC_4	Mgnt_MAC
SRV0255	gfa-sioc-cs-dev	rmgfa-sioc-cs-dev.psi.ch	C*******	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**
SRV0224	gfa-sioc-cs-de4	rmgfa-sioc-cs-de4	C*******	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**	d8:**:**:**:**:**

Abbildung 13: Resultat der generellen CSV-Datei

- **MAC:** Diese Datei stellt alle MAC-Adressen dar, egal ob NIC, Netzwerkadapter oder Devices.

Label	Hostname	Hostname_Mgnt	Mgnt_MAC	NetInterf_MAC_1	NetInterf_MAC_2	NetAdap_MAC_1	NetAdap_MAC_2	NetAdap_MAC_3
SRV0255	gfa-sioc-cs-dev	rmgfa-sioc-cs-dev.psi.ch	d8:*:**:**:**:**	d8:*:**:**:**:**	d8:*:**:**:**:**	d8:*:**:**:**:**	d8:*:**:**:**:**	d8:*:**:**:**:**
SRV0224	gfa-sioc-cs-de4	rmgfa-sioc-cs-de4	d8:*:**:**:**:**	d8:*:**:**:**:**	d8:*:**:**:**:**	-	-	-

Abbildung 14: Resultat der MAC-Adressen CSV-Datei

- **Seriennummern**: Diese Datei stellt alle MAC-Adressen dar, welche gefunden werden konnten (Prozessor, Storage, Memory usw.) Ausserdem enthält sie zusätzlich dazu noch eine kurze Liste mit genaueren Details der Namen und Orte der Seriennummern, um so die einzelnen Teile besser unterscheiden zu können.

Label	Hostname	Hostname_Mgnt	Serial	PowerSupply1	PowerSupply2	Processor1	Processor2	Storage1	Storage2	Memory1	Memory2	Memory3
SRV0255	gfa-sioc-cs-dev	rmgfa-sioc-cs-dev.psi.ch	C*********	5**********	5**********	-	-	S*******************	S**********	-	-	-
SRV0224	gfa-sioc-cs-de4	rmgfa-sioc-cs-de4	C*********	5**********	5**********	-	-	-	-	7*********	7*********	7*********

Additional Information for above

Label	Hostname	Hostname_Mgnt	Serial	PowerSupply_1	PowerSupply_2	Processor_1	Processor_2	Storage_1	Storage_2	Memory_1	Memory_2	Memory_3
SRV0255	gfa-sioc-cs-dev	rmgfa-sioc-cs-dev.psi.ch	C*********	Power Supply 1	Power Supply 2	Intel(R) Xeon(R) CPU E5-269	Intel(R) Xeon(R) CPU E5	HpSmartStorageDiskDriv	HpSmartStor	-	-	-
SRV0224	gfa-sioc-cs-de4	rmgfa-sioc-cs-de4	C*********	HpeServerPowerSupply	HpeServerPowerSupply	Intel(R) Xeon(R) Silver 4416+	Intel(R) Xeon(R) Silver 4	-	-	PROC 1 DIMM	PROC 1 DIMM	PROC 1 DIMM

Abbildung 15: Resultat der Seriennummern CSV-Datei

Überall wo ein ‘-‘ zu sehen ist, wurde kein passender Wert gefunden.

Das in allen drei Dateien vorhandene Feld «Label» ist der Identifier innerhalb des Inventory – es ist daher nur vorhanden, wenn auch eine Inventory-Abfrage durchgeführt wurde und wird aus der Datei ausgelesen, in der das Resultat der Inventory-Abfrage abgespeichert wird.

CSV-Standardisierung
Es gibt bei der Konvertierung von PowerShellvariablen in CSV ein kleines Problem, was ein Array aus Objekten angeht: Es werden nur jeweils die Keys übernommen, welche in allen Instanzen des Objektes vorkommt. Da aber je nach abgefragtem Server nicht überall gleich viele Teile eingebaut; Ein Server hat bspw. 4 RAM-Riegel, ein anderer vielleicht aber 8 oder 16, aber weil ein anderer nur 2 hat, wird in der Datei bei ALLEN nur zwei angezeigt.
Um dieses Problem zu umgehen, kann man mittels einer Funktion über alle Instanzen standardisieren, sodass alle Instanzen die gewünschten Eigenschaften besitzen – dort wo sie vorher nicht vorhanden war, wird sie mit einem ‘-‘ hinzugefügt.

```
Function Get-StandardizedCSV {
    param(
        [Parameter(Mandatory = $true)]
        $Report
    )
    $unique = $Report | ForEach-Object { $_.Keys } | Select-Object -Unique
    foreach ($srvObj in $Report) {
        foreach ($uniqueMember in $unique) {
            if (($srvObj.Keys -contains $uniqueMember) -eq $false) {
                $srvObj | Add-Member -Name $uniqueMember -Value "-" -Member-
Type NoteProperty;
            }
        }
    }
    return $Report
}
```

Codesnippet 20: Standardisierung von CSV-Dateien

2.5.10 Hilfestellungen

Die Hilfestellungen erfolgen pro Funktion jeweils in der Form von «comment-based help» – das ist der native Weg von PowerShell, mit dem üblicherweise dokumentiert wird. Damit lassen sich aber nur die Funktionen dokumentieren, nicht aber das Modul an sich. Aus diesem Grund wurde ein «ILO-Inventorizer.help.txt» erstellt, welches von PowerShell aufgerufen wird, wenn man den «Get-Help»-Befehl auf das Modul anwendet.

Um die Nutzung zu vereinfachen, zeigt die Hilfe des Modules eine kleine Tabelle mit allen Public-Funktionen und jeweils kurz zusammengefasst, was sie so machen. Ausserdem zeigt sie, wie man über die verschiedenen Hilfsparameter zur Hilfe kommt.

```
PS U:\IPA\IPA\hpeilo_inventoryscript\src> Get-Help ILO-Inventorizer
This module provides the functionality not only query hardware information from HPE-ILO machines. It is designed for the Paul Scherrer Institute (PSI) and querries
from the internal database 'inventory.psi.ch'. Its functions are managed via generated config.json-files.

There are a few important functions you should knwo to use:
+--------------------+-------------------------------------------------------------------------------+
| Name               | Function                                                                      |
+--------------------+-------------------------------------------------------------------------------+
| Get-HWInfoFromILO  | Main Function that handles the actual query for ILO and Inventory.            |
|                    | To actually get the Script to do anything (create config, start query etc.) |
|                    | use this function.      |                                                     |
+--------------------+-------------------------------------------------------------------------------+
| Set-ConfigPath     | Set the path to the Config file to somewhere else                             |
+--------------------+-------------------------------------------------------------------------------+
| Get-ConfigPath     | Get the path to the current config file                                       |
+--------------------+-------------------------------------------------------------------------------+
| Get-Config         | Returns the current Config with all its values as a PowerShell-Object        |
+--------------------+-------------------------------------------------------------------------------+
| Update-Config      | Update the current config (use parameters)                                    |
+--------------------+-------------------------------------------------------------------------------+
| Get-NewConfig      | Resets the Current Config and brings up the screen to generate a new one      |
+--------------------+-------------------------------------------------------------------------------+
```

Abbildung 16: Modulhilfe

Die Comment-Based Help ist nur bei den 5 öffentlichen Funktionen zu finden, aus dem Grund, dass die anderen Funktionen gar nicht durch den späteren User verwendet werden sollen und verwendet werden können. Das heisst, um sie zu verwenden, muss man im Code etwas ändern – wenn man so weit ist, sollten die regulären Kommentare beim Programmieren als Hilfestellungen reichen.

2.5.11 Fehlerbehandlung

Die Fehlerbehandlung im Modul geschieht grundsätzlich aus «try-catch»-Blöcken, die wo immer möglich, nicht nur Fehler allgemein abfangen, sondern bei bekannten Fehlern ausserdem den Nutzer darauf hinweisen, wo der Fehler liegt und wie er behoben werden kann.

Dies ist logischerweise nicht abschliessend möglich, da es weitaus mehr Fehler gibt als je während der Realisierung angetroffen wurden. Es wurde nichtsdestotrotz darauf geachtet, die Parameter und Konfigurationsdateien möglichst breit und feinkörnig abzudecken. Durch die Möglichkeit in PowerShell mit Catch auch spezifische Fehler zu filtern und sie gesondert zu behandeln hat teils sehr geholfen.

```
try {
    # Return Inventorydata from previously saved file.
    $config = Get-Config;
    $path = $config.searchForFilesAt + "\inventory_results.json";
    Log 5 "Import Inventory Data from '$path'"
    if (Test-Path -Path $path) {
        $server = Get-Content $path | ConvertFrom-Json -Depth 10;
    }
}
catch [System.IO.FileNotFoundException], [System.IO.DirectoryNotFoundExcep-
tion] {
    Save-Exception $_ "The file $path does not exist or has been moved. Do not
move or delete, as it is vital to query from Inventory.";
}
```

Codesnippet 21: Fehlerbehandlung mit verschiedenen Catches

Zudem wurde die oben sichtbare Funktion **Save-Exception** implementiert, um das Handling zu vereinfachen – durch sie wird die Speicherung des Fehlers und dessen Ort zentralisiert, was einige Zeilen Code spart.

```
Function Save-Exception {
    param(
        [Parameter(Mandatory = $true)]
        $_,

        [Parameter(Mandatory = $true)]
        $Message
    )
    # Save Exceptions to Logs and log them to the console
    Log 1 ("$Message`n" + $_.ScriptStackTrace);
    Write-Error ("$Message");
}
```

Codesnippet 22: Save-Exception Funktion

Speziell musste die *Connect-HPEILO* Funktion von der Bibliothek behandelt werden, da dort nicht reguläre Fehler geworfen werden, sondern diese komplett anders aussehen. Dort musste deswegen mit **-match**-Statements gearbeitet werden.

```
catch {
    $message = $_.Exception.Message.ToString();
    if ($message -match "401") {
        Save-Exception $_ "The ILO-Server $srv returned Unauthorized. […]"
        }
    elseif ($message -match "SSL") {
        Save-Exception $_ "The ILO-Server $srv returned an Error with SSL.
[…]";
        }
        else {
            Save-Exception $_ ($_.Exception.Message.ToString());
        }
    }
```

Codesnippet 23: Fehlerbehandlung Connect-HPEILO

Um die Validität der Konfiguration zu garantieren, wird beim Auslesen der Konfiguration jeweils geprüft, ob der jeweilige Typ stimmt. Genauere Validierungen werden dann erst bei der Verwendung der einzelnen Eigenschaften genauer überprüft (z.B. ob Pfad existiert)

```
if (
    ($config.searchForFilesAt -isnot [string]) -or
    ($config.configPath -isnot [string]) -or
    ($config.loginConfigPath -isnot [string]) -or
    ($config.reportPath -isnot [string]) -or
    ($config.serverPath -isnot [string] -and ($null -ne $config.serverPath)) -
or
    ($config.logLevel -isnot [int64]) -or
    ($config.searchStringInventory -isnot [string] -and ($null -ne $con-
fig.searchStringInventory)) -or
    ($config.remoteMgmntField -isnot [string]) -or
    ($config.LoggingActivated -isnot [bool]) -or
    ($config.doNotSearchInventory -isnot [bool]) -or
    ($config.deactivateCertificateValidation -isnot [bool]) -or
    ($config.logToConsole -isnot [bool]) -or
    ($config.ignoreMACAddress -isnot [bool]) -or
    ($config.ignoreSerialNumbers -isnot [bool]) -or
    ($config.deactivatePingtest -isnot [bool])
) {
throw [System.IO.InvalidDataException] "Your configuration has wrong types.
Please verify that the following types are met:`nlogLevel = int, LoggingAc-
tivated = bool, doNotSearchInventory = bool, deactivateCertificateValidation =
bool, logToConsole = bool, ignoreMACAddress = bool, ignoreSerialNumbers =
bool, deactivatePingtest = bool AND any other fields are of type string."
}
```

Codesnippet 24: Validation Konfiguration

Eine weitere Massnahme der Fehlerbehandlung ist auch die Validierung der Pfade aus der Konfiguration. Diese werden im Falle von Pfaden zu Ordnern erstellt, falls sie nicht existieren. Falls sie aber Pfade zu Dateien sind können sie leider nicht automatisch erstellt werden, da sich nicht auf den möglichen Inhalt schliessen lässt. Dies wurde als Funktion Convert-PathsToValidated implementiert.

```
$config = Get-Config;

# searchForFilesAt-Property
if (-not(Test-Path -Path ($config.searchForFilesAt))) {
    Log 6 ("`tCreating Path 'searchForFilesAt: " + $config.searchForFilesAt);
    New-Item -ItemType Directory ($config.searchForFilesAt) -Force | Out-Null;
        }

# loginConfigPath-Property --> Error because path must be to a file -> the
contents of which cannot be generated automatically
if ((-not(Test-Path -Path ($config.loginConfigPath)))) {
   Log 6 ("`tPath 'loginConfigPath' does not exist: " + $config.loginCon-
figPath);
   throw [System.IO.FileNotFoundException] ("Path to '$path' could not be re-
solved. Verify that loginConfigPath includes some file like 'login.json' and
it and the file must exist for the script to work. It also must include a
Username and a Password.")
}
```

Codesnippet 25: Ausschnitt Convert-PathsToValidated

2.5.12 Sicherung des Passwortes

Passwörter sind sehr oft etwas Empfindliches, womit sich viel Schaden anrichten lässt. Deswegen ist es üblich, das Passwort zu verschlüsseln. In PowerShell kann das über Secure-Strings gelöst werden. In dem Fall dieser IPA gibt es zwei Wege, wie das Passwort gesetzt werden kann – entweder in einer Konfigurationsdatei (separat zur normalen) oder per Parameter.

Um die Sicherheit zu gewährleisten wird deswegen intern das Passwort immer als Secure-String gehandelt; Bei dem Start per Parameter ist die Eingabe in Form eines Secure-Strings, beim Auslesen aus einer Datei wird es von PlainText in einen SecureString konvertiert. D.h. sobald das Passwort sich im Scope des Moduls befindet, wird es sicher behandelt.
Das Passwort selbst ist nur relevant für die Anmeldung am Server, weshalb es dort wieder als PlainText zurückkonvertiert wird.

```
$conn = Connect-HPEiLO -Address $srv -Username $login.Username -Password (ConvertFrom-SecureString -SecureString ($login.Password) -AsPlainText)
```

Codesnippet 26: Implementation Verbindungsaufbau

Die Speicherung als SecureString ist aber im Projektumfeld nicht unbedingt erforderlich, aus folgenden Gründen:

- Ist man am Server, kann man direkt daran das Root-Passwort ablesen und kommt so ohne weiteres an das ILO-Interface,
- Ist man wie am PSI in mehreren verschachtelten und abgesicherten Netzwerken, ist die individuelle Sicherheit des Passworts weniger wichtig. Die Server, welche abgefragt werden, befinden sich so tief in gesicherten Netzwerken, sodass man auf dem Weg dahin mehrmals Passwörter eingeben muss, eine Zwei-Faktor-Authentifizierung durchgeführt wird und man einige Male das Netzwerk wechseln muss. Ist man also im Netzwerk, in dem sich die knapp 42 Server befinden, so ist man schon auf einem solchen Level an Sicherheit, dass alle Maschinen das gleiche standardisierte Passwort und denselben Benutzernamen haben.

Das Script wird später als Service laufen, wobei dafür im Active-Directory das Login gespeichert werden könnte – in diesem Falle ist das Handling als SecureString sinnvoll, da es dort eine Schnittstelle gibt, durch die angegriffen werden könnte.

2.6 Kontrollieren

2.6.1 Testprotokoll 1 - 19.03.2025

Datum	19.03.2025
Tester	Yannick Wernle
Ergebnis	Zufriedenstellend

Testfälle

Testnr.	Testfallnr.	Resultat	Bemerkung
01	TF-01	OK	-
02	TF-02	OK	Fehler wird geworfen, Beschreibung könnte besser sein
03	TF-03	NOK	Es wird zwar ein Fehler geworfen, jedoch nicht wegen des Typs, sondern aufgrund eines Loops im Logfile --> wurde angepasst und wird nun abgefangen
04	TF-03	OK	
05	TF-04	NOK	Fehler wird geworfen, dass ein Parameter namens '-' nicht existiert --> Übersicht im Code, wurde entfernt
06	TF-04	OK	
	TF-05	NOK	Fehler wird geworfen beim Erstellen des Dummy-Logpfades, da eine Referenz auf einen Ordner gemacht wird, der nicht existiert -> wurde entfernt
07	TF-05	OK	
08	TF-06	OK	
09	TF-07	OK	
10	TF-08	NOK	Property 'Serial' does not exist -> es lag an einem Fehler beim Abfangen der ILO-Version -> wurde nun behoben
11	TF-08	OK	
12	TF-09	OK	
13	TF-10	OK	
14	TF-11	NOK	Konfigurationsdatei wird nicht richtig generiert --> habe es angepasst und wird wieder richtig konfiguriert.
15	TF-11	OK	
16	TF-12	OK	
17	TF-13	OK	
18	TF-14	NOK	Konfiguration wurde überschrieben und nicht korrekt gespeichert. --> wurde angepasst.

19	TF-14	NOK	Pfad zur Reportdatei zeigte noch auf einen Pfad der nicht existiert --> behoben
20	TF-14	OK	
21	TF-15	OK	
22	TF-16	OK	
23	TF-17	NOK	ignoreMACAddresse wurde durch Update-Config immer wieder überschrieben --> behoben
24	TF-17	OK	
25	TF-18	OK	

Testbericht

Wie oben zu sehen ist, gab es währen den ganzen Tests einige Fehler, die noch ausgebügelt werden mussten. Die Testfälle wurden soweit abgedeckt, dass alle individuell laufen sollten. Die Applikation ist im gefixten Zustand bereits weitgehend lauffähig. Schlecht ist sicher, dass es noch so viele Probleme gab, die eigentlich hätten abgefangen werden sollen – das ist bei IPERKA eigentlich nicht so gedacht.

Auf Vorschlag des VFs konnte das Script in einer «scharfen Umgebung getestet werden. Da dies nach den oben aufgeführten Tests geschah, gelang dies (trotz einiger kleiner Fehler) relativ schnell und am Ende konnte ausserdem wie gewünscht alle CSV & JSON generiert werden. D.h. das Script funktioniert und wäre in diesem Zustand bereits auslieferfertig. Da der VF aber auch noch 2-3 kleinere Anmerkungen und Bugs hatte, mussten diese noch nachträglich behoben werden (Erstellung von Pfaden, wenn sie nicht bereits existieren z.B.)

Das Resultat des scharfen Testes wurde dann an einen der späteren Benutzer (Kurt Bitterli) geschickt, um Feedback zu erhalten – laut ihm sind alle wichtigen Daten, die er benötigt, darin vorhanden.

Wie der Testlauf aber auch gezeigt hat, gibt es vor allem im Bezug auf die Fehler, welche abgefangen werden, noch Verbesserungsbedarf, der aber aufgrund der kurzen Zeit der IPA auch zu erwarten ist.

2.7 Auswerten

2.7.1 Technisches Fazit

Der Grossteil aller Anforderungen und Funktionen konnte gemäss Aufgabenstellung implementiert werden: Das Programm lässt sich ohne Probleme Starten und Konfigurieren und gibt gemäss Testprotokoll auch das Gewollte zurück, d.h. die wichtigsten Funktionen wurden implementiert und das Programm ist auslieferbar.

Mängel und Probleme gibt es vor allem noch im Bereich der Fehlerbehandlung, wo es noch Optimierungsbedarf gibt – es werden zwar bereits viele Fehler abgefangen und versucht überall, wo möglich dem Anwender den Fehler mitzuteilen und wie er gefixt werden kann, jedoch ist das nicht überall gelungen, insbesondere bei der Anforderung, dass bei fehlerhaften Parametern in den Funktionen Fehler angezeigt werden und wie sie gefixt werden. Es gibt da zwar bereits implementierte Teile wie die Überprüfung des Typs oder ob die Überprüfung ob Pfade existieren, das könnte und sollte aber noch spezifischer auf die einzelnen Parameter angewendet werden. Anstatt diese Werte wie implementiert dezentral zu überprüfen wäre es geeigneter, sie zentral zu validieren, um so genauere Fehler abfangen zu können.

Es wurde sich, wo immer möglich, an Best-Practices und Vorlagen gehalten: So sind zum Beispiel alle Parameter jeweils im Pascal Case (**P**ascal**C**ase) und die restlichen Variablen im Camel Case (**c**amel**C**ase) geschrieben. Weiter wurden try-catch für die Fehlerbehandlung verwendet und zu allen öffentlichen Funktionen kommentarbasierte Hilfen von PowerShell implementiert. Ausserdem wurde die Struktur des Programmes als Modul implementiert, welches ebenfalls einem PowerShell-Standard folgt. Aufgrund dieser Struktur liess sich auch das Konzept «Keep it simple and stupid» gut umsetzen, da ohne Probleme verschiedene Dateien und Funktionen eingesetzt werden konnten.

Diese Kommentarbasierte Hilfe trägt unter anderem auch zur Dokumentation des Programmes bei. Ergänzend dazu wurden noch reguläre Kommentare verwendet, sowie die implementierte Logfunktion, welche auch zur Dokumentierung beiträgt.
Insgesamt funktioniert das Programm gut – es wurde auf einer «scharfen» Maschine, welche sich in der Grossforschungsanlage SwissFEL befindet, getestet und konnte erfolgreich seinen Zweck erfüllen.

2.7.2 Zukunftsaussichten

Verbesserungspotential gibt es sicher im Bereich der Fehlerbehandlung: Die oben benannte Hilfe bei falschen Parametern sollte noch spezifischer und vor allem zentral gemacht werden.

Das Programm kann ausserdem auf andere Server ausgeweitet und getestet werden, welche nicht den Grenzen der IPA unterliegen: So z.B. die DL20-Systeme, welche ebenfalls noch am PSI Verwendung finden.

Weitere mögliche Features und Eigenschaften, welche implementiert werden könnten:
- Deployment in ein Repository wie www.powershellgallery.com,
- CI/CD im Git-Repository zum Automatisieren des Testings und des Deployments,
- Implementation von Unittests mit einem Framework wie Pester, um das Testen massiv zu vereinfachen und zu verkürzen,
- Anpassung des Outputs in Absprache mit Alain Bertrand, sodass das JSON direkt wieder zum Import ins Inventory verwendet werden könnte.
- Abbildung der Baumstruktur wie sie im Inventory existiert im Output.
- Absprache mit Kurt Bitterli bezgl. der Struktur und Optimierung der Dateiausgabe

2.7.3 Selbstreflexion & Persönliches Fazit

Ich bin mit dem Zustand des Produkts und der IPA allgemein zufrieden – ich konnte, soweit ich das beurteilen kann, die Aufgabenstellung umsetzen, ohne dass etwas grosses und/oder wichtiges fehlt.

Das Implementieren im Allgemeinen konnte ich ohne gravierende Probleme umsetzen und ich wurde nie durch grosse Probleme in Zeitdruck gebracht. Die Vorbereitung auf die IPA in Form einer Test-IPA hat mir sehr geholfen – nicht nur um mich wieder (nach 3 Jahren Pause) an PowerShell zu gewöhnen, sondern auch um die HPEiLOCmdlets-Bibliothek kennenzulernen. Ich war dadurch gut vorbereitet, sodass ich ohne gross Nervosität in die IPA starten konnte und während der ganzen IPA, ausser während des Testens, sehr ruhig und fokussiert an die Aufgabenstellung herangehen konnte.

Schwierig fand ich vor allem, die Aufgabenstellung im Auge zu behalten und alles zu berücksichtigen, was gefordert ist: Ich habe ein- zweimal beim Überprüfen der Aufgabenstellung bemerkt, dass mir noch Teile davon fehlen – diese musste ich dann noch nachimplementieren. Ich habe zwar Github-Issues verwendet, um dieses Problem zu vermeiden, aber wie sich gezeigt hat, waren sie nicht detailliert genug. Das kann und will ich für die Zukunft noch verbessern – ausserdem haben mir die Issues insgesamt auch schon im jetzigen Zustand gut aufgezeigt, wie weit ich mit dem Programm war.

Ich habe während dem Testen schmerzlich merken müssen, wie wichtig und nützlich Unittests gegenüber von Usertests zur Laufzeit sein können – ich hatte im Vorfeld der Testphase eigentlich gedacht, dass ich einen Grossteil der Tests ohne Probleme abschliessen kann, aber dem war nicht so – ich musste während dem Testen noch eine gefühlte Flut an Fehlern beheben, damit die Tests erfolgreich verliefen. Das hat mir unter anderem auch wieder die Schwächen von Wasserfallmodellen bei der Organisation von Informatikprojekten gezeigt – ich war dem aber schon bewusst und habe sie bewusst in Kauf genommen. Diese Fehler hätten durch Unittests während der Implementierung behoben werden können, ohne dass dies zu viel Aufwand geführt hätte. Deswegen werde ich bei der Weiterentwicklung des Scriptes nach Abschluss der IPA Unittests implementieren, um Fehler dieser Art schneller und einfacher erkennen und beheben zu können.

Ich wusste aus früheren Projekten, wie viel Arbeit es gibt, wenn man im Nachhinein Code noch kommentieren und dokumentieren muss – das konnte ich unter anderem durch die Logfunktion und den Puffer am Ende der IPA aber problemlos und relativ ausführlich machen – auch die Hilfestellungen des Moduls (Kommentarbasierte Hilfe) und die Fehlerbehandlung tragen dazu bei und ich finde, das ist mir dieses Mal wesentlich besser gelungen. Ich habe ausserdem auch darauf geachtet, Coding-Konventionen so gut es ging zu beachten, sodass ich am Ende nicht noch viele Variablen und Funktionen anpassen musste.

Der Zeitplan ist mir meiner Meinung nach auch gut gelungen: Der Ansatz aus der Test-IPA, die einzelnen Phasen so kleinkörnig wie möglich aufzuführen hat sich als gut erwiesen – ich konnte sehr gut erkennen, wenn ich vor- oder hinter dem Zeitplan war und konnte dementsprechend schnell reagieren.

Die IPA war nicht nur bloss ein Test, sondern hat auch Spass gemacht, besonders als ich die Resultate des Tests auf der «scharfen» SwissFEL-Maschine gesehen habe.

3 Literaturverzeichnis

[deleted], R. . (2022, 9 23). *At what point does a script become a module*. Retrieved from Reddit: https://www.reddit.com/r/PowerShell/comments/xm4ryo/at_what_point_does_a_script_become_a_module/?rdt=61937

codeConcussion. (2013, 04 04). *Select-Object on nested collections*. Retrieved from stackoverflow: https://stackoverflow.com/questions/15816491/select-object-on-nested-collections

DarkLite1. (2014, 07 15). *PowerShell Remove item [0] from an array*. Retrieved from stackoverflow: https://stackoverflow.com/questions/24754822/powershell-remove-item-0-from-an-array

gskinner. (n.d.). *PowerShell Regex*. Retrieved from Regexr: PowerShell Regex: https://regexr.com/3c0lf

Hewlett Packard Enterprise. (n.d.). *Was ist Integrated Lights-Out (iLO)?* Retrieved from Hewlett Packard Enterprise: https://www.hpe.com/ch/de/what-is/ilo.html

JaapBrasser. (2019, 06 28). *stackoverflow*. Retrieved from How to check if a PowerShell switch parameter is absent or false: https://stackoverflow.com/questions/56809557/how-to-check-if-a-powershell-switch-parameter-is-absent-or-false

jeremy.hawkins1. (2023, 05 01). *Export-Csv array of objects Error " Object reference not set to an instance of an object.".* Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/answers/questions/1185618/export-csv-array-of-objects-error-object-reference

Kaarsemaker, D. (2021). *iLO automation from python or shell¶*. Retrieved from python-hpilo: https://seveas.github.io/python-hpilo/index.html

Maclean, W. (2018, 11 22). *Tracing where output is coming from in a Powershell script*. Retrieved from stackoverflow: https://stackoverflow.com/questions/53423718/tracing-where-output-is-coming-from-in-a-powershell-script

Marquette, K. (2017, 03 27). *Powershell: Building a Module, one microstep at a time*. Retrieved from PowerShell Explained: https://powershellexplained.com/2017-05-27-Powershell-module-building-basics/#psm1

Microsoft. (n.d.). *ConvertFrom-SecureString*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/convertfrom-securestring?view=powershell-7.5

Microsoft Learn. (2021, 09 18). *Understanding a Windows PowerShell Module*. Retrieved from Microsoft | Learn | PowerShell: https://learn.microsoft.com/en-us/powershell/scripting/developer/module/understanding-a-windows-powershell-module?view=powershell-7.5

Oliver, S. . (2011, 02 24). *When to choose development of a PowerShell Module over PowerShell Script*. Retrieved from stackoverflow: https://stackoverflow.com/questions/5103211/when-to-choose-development-of-a-powershell-module-over-powershell-script

Pester. (n.d.). *Quick Start*. Retrieved from Pester.dev: https://pester.dev/docs/quick-start

pl5. (2024, 01 18). *HPEiLOCmdlets 4.3.0.0 Error when using Get-HPEiLOIML*. Retrieved from Hewlett Packard Enterprise: https://community.hpe.com/t5/proliant-servers-ml-dl-sl/hpeilocmdlets-4-3-0-0-error-when-using-get-hpeiloiml/td-p/7204825

Powell, D. (2012, 06 07). *In PowerShell, how can I test if a variable holds a numeric value?* Retrieved from stackoverflow: https://stackoverflow.com/questions/10928030/in-powershell-how-can-i-test-if-a-variable-holds-a-numeric-value

Scripto, D. (2014, 08 31). *PowerTip: Using PowerShell to Determine if Path Is to File or Folder*. Retrieved from Microsoft | Devblogs: https://devblogs.microsoft.com/scripting/powertip-using-powershell-to-determine-if-path-is-to-file-or-folder/

sdwheeler. (2024, 06 05). *about_Special_Characters*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_special_characters?view=powershell-7.5
sdwheeler, A. (2024, 01 19). *about_Requires*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_requires?view=powershell-7.5
sdwheeler, A. (2025, 01 10). *about_Comment_Based_Help*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_comment_based_help?view=powershell-7.5
sdwheeler, ArieHein, sethvs, Blake-Madden. (2025, 01 31). *about_Regular_Expressions*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_regular_expressions?view=powershell-7.5
sdwheeler, c.-h. T.-S. (2024, 10 17). *Approved Verbs for PowerShell Commands*. Retrieved from Microsoft | Learn: https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands?view=powershell-7.5
tnw. (2013, 09 18). *PowerShell and the -Contains operator*. Retrieved from stackoverflow: https://stackoverflow.com/questions/18877580/powershell-and-the-contains-operator
Wikipedia. (2025, 01 21). *HP Integrated Lights-Out*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/HP_Integrated_Lights-Out
Wikipedia. (2025, 03 08). *Out-of-band management*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Out-of-band_management

4 Abbildungsverzeichnis

Abbildung 1: Beispiel Abbildung	6
Abbildung 2: Organigramm	11
Abbildung 3: Ausschnitt Github Commits	12
Abbildung 4: Git Issues	13
Abbildung 5: Arbeitstage	15
Abbildung 6: Projektumfeld	34
Abbildung 7: Programmablaufplan (simpel)	37
Abbildung 8: Gesonderte Netzwerkschnittstelle von ILO	38
Abbildung 9: ILO-Chip in einem ProLiant Gen9	38
Abbildung 10: Informationsansicht Inventory	39
Abbildung 11: Programmablaufplan Script detailliert	52
Abbildung 12: Ordnerstruktur	54
Abbildung 13: Resultat der generellen CSV-Datei	64
Abbildung 14: Resultat der MAC-Adressen CSV-Datei	64

Abbildung 15: Resultat der Seriennummern CSV-Datei 64
Abbildung 16: Modulhilfe 65

5 Tabellenverzeichnis

Tabelle 1: Versionierung 1
Tabelle 2: Verteiler 2
Tabelle 3: Dokumentinformationen 2
Tabelle 4: Beispieltabelle 6
Tabelle 5: Zeitplanung 15
Tabelle 6: Arbeitsjournal Tag 1 18
Tabelle 7: Arbeitsjournal Tag 2 19
Tabelle 8: Arbeitsjournal Tag 3 20
Tabelle 9: Arbeitsjournal Tag 4 21
Tabelle 10: Arbeitsjournal Tag 5 22
Tabelle 11: Arbeitsjournal Tag 6 23
Tabelle 12: Arbeitsjournal Tag 7 25
Tabelle 13: Arbeitsjournal Tag 8 27
Tabelle 14: Arbeitsjournal Tag 9 29
Tabelle 15: Arbeitsjournal Tag 10 31
Tabelle 16: Arbeitsjournal Tag 11 31
Tabelle 17: Arbeitsjournal Tag 12 32
Tabelle 18: Entscheidungsmatrix Programmierstruktur 53

6 Codesnippetverzeichnis

Codesnippet 1: Inhalt der Server-Konfiguration	49
Codesnippet 2: Geplante Parameter	51
Codesnippet 3: Inhalt des Startscripts	54
Codesnippet 4: Get-Help Funktionalität	55
Codesnippet 5: Überprüfung Modulversion	55
Codesnippet 6: "Frontend" Generierung der Konfigurationsdateien	56
Codesnippet 7: ParameterSet am Benutzer	57
Codesnippet 8: Syntax Get-HWInfofromILO	57
Codesnippet 9: Beispiel Update-Config	57
Codesnippet 10: Ausschnitt Logdatei	59
Codesnippet 11: Aufruf Pingtest	59
Codesnippet 12: Abfrage Inventory	60
Codesnippet 13: Konvertierung der Inventory-Antwort	60
Codesnippet 14: ILO-Abfrage & Standardisierung von Memory	61
Codesnippet 15: Modifizierte Logfunktion	62
Codesnippet 16: DeviceInventory Inkompatibilität	62
Codesnippet 17: Ausgabe bei Inkompatibilität	62
Codesnippet 18: Dateispeicherung JSON	63
Codesnippet 19: Generierung des Pfades falls er nicht existiert.	63
Codesnippet 20: Standardisierung von CSV-Dateien	65
Codesnippet 21: Fehlerbehandlung mit verschiedenen Catches	66
Codesnippet 22: Save-Exception Funktion	66
Codesnippet 23: Fehlerbehandlung Connect-HPEILO	67
Codesnippet 24: Validation Konfiguration	67
Codesnippet 25: Ausschnitt Convert-PathsToValidated	68
Codesnippet 26: Implementation Verbindungsaufbau	69

7 Glossar

Fachbegriff	Erläuterung
CSV	CSV (siehe Abkürzungsverzeichnis) bezeichnet den Aufbau einer Textdatei, in der verschiedene Werte durch die im Namen enthaltene Kommas getrennt wird. Sie sind aber nicht spezifisch auf Kommas eingeschränkt, es können auch z.B. Semikolon als Trenner verwendet werden. ➔ Bspw. Name; Address DevServer; rmgfa-sioc-cs-dev; DevServer3; rmgfa-sioc-cs-de3 ;
HPEiLOCmdlets	Bibliothek/Cmdlet für PowerShell, mit dem ILO konfiguriert und abgefragt werden kann.
HTTPS	«Hypertext Transfer Protocol Secure» Netzwerkprotokoll zur verschlüsselten Datenübertragung im Internet – sichere Weiterentwicklung von HTTP.
ILO	ILO (siehe Abkürzungsverzeichnis) ist ein von Hewlett Packard Enterprise entwickeltes System zur Wartung und Administration von Servern, für welches man nicht in der Nähe des Servers sein muss, sondern sich remote darauf verbindet.
Inventory.psi.ch	Interne Datenbank des PSI zur Erfassung und Wartung von Elektronischer und informationstechnischen Gegenständen, Teilen und Servern.
IPERKA	IPERKA ist eine Projektmanagementmethode, welche als Wasserfallmodell, bestehend aus den Phasen Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Auswerten.
JSON	JSON (siehe Abkürzungsverzeichnis) bezeichnet eine Art der Notation von Objekten. Sie ist simpel und wird bei vielen Webanwendungen verwendet.
Maschinennetz	Geschütztes und abgesichertes Netzwerk ohne Internetzugang, in dem sich die Forschungscomputer und Forschungsdaten befinden.
Officenetz	Geschütztes Netz am PSI, in welchem sich die normalen Rechner befinden. Es ermöglicht Zugang auf interne Services wie Inventory.psi.ch oder SAP, aber nicht auf Server und Forschungsdaten und enthält Zugang zum Internet.
ParameterSet	Feature von PowerShell, mit dem innerhalb einer einzigen Funktion zwischen verschiedenen Szenarien von Parametern unterschieden werden kann.
Pester	Framework für PowerShell, mit dem Unittests geschrieben und automatisch ausgeführt werden können.

8 Abkürzungsverzeichnis

Abkürzung	Erläuterung
CSV	Abkürzung für «Comma Separated Values»
HEX	Abkürzung für «Hauptexperte»
ILO	Abkürzung für «Integrated Lights Out”
JSON	Abkürzung für «JavaScript Object Notation»
KAND	Abkürzung für «Kandidat» - dies ist der Autor des IPA-Berichts.
NEX	Abkürzung für «Nebenexperte
NIC	Abkürzung für «Network Interface Card» - dies bezeichnet die verbaute Netzwerkkarte(n)
PAP	Abkürzung für «Programmablaufplan»
PSI	Abkürzung des Lehrbetriebs «Paul Scherrer Institut»
VF	Abkürzung für «Verantwortliche Fachkraft»

9 Selbstständigkeitserklärung

Hiermit erkläre ich, Yannick Wernle, dass ich die vorliegende IPA-Arbeit selbstständig und ohne fremde Hilfe verfasst habe. Sämtliche verwendeten Quellen und Hilfsmittel sind im Bericht vollständig und korrekt angegeben. Ich bestätige zudem, dass alle KI-Anfragen, die während der Erstellung dieser Arbeit gemacht wurden, im Bericht dokumentiert sind.

Ort, Datum: 24.03.2025
Unterschrift: Y. Wernle

10 Anhang

10.1 Protokoll Erster Expertenbesuch

Datum	Teilnehmer	Dauer
05.03.2025	HEX, VF, KAND	Ca. 45 min

Traktanden:
- Kurze Vorstellung,
- Besprechung Aufgabenstellung
- Zeitplan anschauen & Stand der Dokumentation überprüfen,

Bemerkungen HEX & VF:
- Bei Glossar auf alphabetische Reihenfolge achten,
- Bei der Projektorganisation begründen, wieso IPERKA und nicht einfach «weil ich es schon verwende»
- Bei der Entscheidungsphase nicht etwas nehmen wo man schon im Vorherein weiss, was man nimmt.
- Quellenangaben und KI sind erlaubt und es reicht einfach ein Link, resp. zu erwähnen, wie man es verwendet hat, aber eine konkrete Angabe des Prompts ist nicht nötig.
- (VF) Beim PAP sollte die Versionsüberprüfung am Anfang stattfinden.

Fragen:
- Was abgeben? --> Falls im Git schon alles vorhanden ist, dann reicht dass, ansonsten der Rest dazu (Dokumentation, Zeitplan usw.)

10.2 Protokoll Zweiter Expertenbesuch

Datum	Teilnehmer	Dauer
18.03.2025	HEX, VF, KAND	Ca. 43 min

Traktanden:
- Kurze Besprechung über Stand
  - Zeitplan angeschaut
  - Dokumentation angeschaut
  - Kurzes Code Review
- Information über Präsentation & nachfolgendes Fachgespräch

Bemerkungen HEX & VF:
- Speicherung des Passwortes in einem normalen String ist nicht gut – wieso wird nicht ein SecureString verwendet?
- Grafiken & Code in der Präsentation sollten gross, scharf und gut lesbar sein (ansonsten drucken)
- Dem HEX ist auch der Kontext rund um das Projekt wichtig (Wer involviert, Wieso braucht es usw.), das Inhaltsverzeichnis sollte so kurz wie möglich sein
- VF: Klar erwähnen, dass die Modulversion und ILO-Abfrage bereits Teil der Test-IPA war.
- VF: Ich soll bei den Variablennamen die Formatierung so einheitlich als möglich machen und die Parameter klar als Usereingaben kennzeichenn.
- Am Besten den Montag grösstenteils in die Dokumentation einfliessen lassen und bereits vorher schon etwas hochladen aber nicht signieren (falls was passieren würde). Der Abgabezeitpunkt muss nicht 1 zu 1 um 13:00 Uhr sein, paar Sekunden vorher oder nachher ist auch okay.

Fragen:
- Abgabe der Dokumentation sollte vor allem als PDF sein oder?
  Genau --> kann auch im Ordner als .docx abelegt sein, aber PDF garantiert die einheitliche Darstellung, welche bei Word nicht garantiert ist.