Inhaltsverzeichnis

1 Automatisierung

1.1 Einführung

- Arbeit handelt um Automatisierung
- Warum wurde dieses Thema für die Arbeit gewählt?
- Automatisierung von Programminstallationen
- Betreuung und Analyse eines gesamten zu Automatisierenden Prozesses
- Daraufsicht auf den Prozess der Automatisierung managen und arbeiten nach ISO bzw. ITIL Konformität Praxis 4.

Häufig gibt es gewisse Aufgaben oder Prozesse, welche keine großen Unterschiede in der Abwicklung aufweisen. Für jene kann sich mit der Anwendung von Automatisierung beschäftigt werden. Die generelle Anforderung an Automatisierung ist es Arbeits- oder Produktionsprozesse für den Menschen so durchzuführen, dass dieser nicht unmittelbar tätig werden muss, um Aufgaben zu verrichten. Spezifischer auf das Themenfeld der Projektarbeit bezogen, stellt sich die Aufgabe dar, mit Hilfe von Skripten Installationsprozesse auf ein Minimum von Zeitaufwand zu beschränken. https://m.bpb.de/nachschlagen/lexika/lexikon-der-wirtschaft/18743/automatisierung Das Skript soll also insoweit die Prozesse automatisieren, dass vom Nutzer lediglich ein paar Parameter angepasst werden müssen, die sich bei jeder Installation unterscheiden

Das Ziel der Arbeit ist es also neben der klassischen Dokumentierung, was in den Praxisphasen erarbeitet wurde, einen guten Überblick über die Herangehensweise, Analyse und das Potential von Automatisierung zu verschaffen.

2 Task Automation mit Powershell

Der Schwerpunkt dieser Praxisarbeit wurde auf die Implementierung von zu automatisierenden Prozessen in PowerShell gelegt. Es wird beginnend bei der Ideenfindung für einen solchen Prozess, bis zur abschließenden Einbindung, ein umfassender Überblick gegeben, wie Projekt und Aufgabe abgelaufen sind.

2.1 Aufgabenstellung

Als äußerste Frage, welche es zunächst zu beantworten galt, wurde die Folgende formuliert:

Wie können Prozesse mit Hilfe von Powershell weitestgehend automatisiert werden?

> Aus jener Frage lassen sich mehrere Aufgaben ableiten. Einerseits kristallisiert sich daraus das Recherchieren und Verstehen von Prozessen, sowohl im übergeordneten Zusammenhang mit Anderen, als auch der Eigentliche an sich. Geht man beispielsweise von einer Programm Installation aus ist dies der Kernprozess, er hängt aber übergeordnet mit den Berechtigungen die auf dem Betriebssystem gelten zusammen. Andererseits spielt die Aufgabe des strukturierten Dokumentieren und Vorgehen bei der Umsetzung eine große Rolle.

Nicht zuletzt muss sich mit der Verwendung von Powershell, vielmehr der Implementierung in der Powershell ISE, befasst werden. Auf die Aufgabe des Implementierens wurde in diesem Teil der Arbeit das Hauptaugenmerk gelegt.

Ziel der Aufgabe ist es also den gewählten Prozess weitestgehend zu automatisieren. Durch die implementierten Skripte soll dabei eine schnellere und quali-

tativ gleichbleibende Installation, sowie Konfiguration der Software ermöglicht werden.

2.2 Einordnung der Aufgabenstellung in übergeordnete Prozesse

Ausgewählt wurde diese Aufgabe in der Abteilung Virtualisierung, da bei Digitalen Arbeitsplätzen die Automatisierung solcher Prozesse eine generell recht übliche Prozedur ist. Zudem ist das Aufsetzen und die Verwaltung, insbesondere von virtuellen Desktops, meist nur minimal Abweichend, wenn nicht sogar nahezu identisch in der Durchführung.

Demnach kann das Unternehmen durch die Automatisierung kosten sparen und bietet zudem die Verlagerung von Ressourcen. Ganzheitlich gesehen können durch die gewonnene zeitliche Ersparnis, die Arbeitskräfte in die Umsetzung komplexerer Aufgaben gesetzt werden.

Arbeiten im AD soll vereinfacht werden, dass der Nutzer nicht mehr auf einzelne Schritte zurückgreifen muss.

2.3 Praktische Umsetzung

Zunächst soll im Analyseteil der Aufgabe ein grundlegendes Verständnis zu den ablaufenden Prozessen aufgebaut werden. Hierfür wurde eine kurze Liste mit den in der Abteilung verwendeten Produkten ausgehändigt:

- Citrix Cloud Connector (CC)
- Powershell
- PowerShell ISE

Festgelegt wurde sich auf die Installations Automatisierung des Citrix Cloud

Connectors.

In den folgenden Unterkapiteln soll ein genereller Überblick über die in dem Projekt verwendete Software geschaffen werden. Dabei soll dem Leser ein klarer Nutzen und Mehrwert des Produkts, sowie eine kurze Heranführung über die verwendeten Funktionen, in dem Projekt, erläutert werden.

2.3.1 Citrix Cloud Connector

Der Citrix Cloud Connector ist ein Produkt, das die Verbindung zu einem Citrix Cloud-Server ermöglicht. Dieser kann wiederum einen virtuellen Desktop hosten und gewährt den Zugriff von anderen Computern auf diese Virtuelle Maschine. Dabei sei angemerkt, dass von dem Nutzer auch eigene Ressourcenstandorte auf Serverseite integriert werden können. Der Vorteil dabei ist, das von überall mit nahezu jeder Art von Hardware ein solches System genutzt werden kann. Die Erreichbarkeit durch den Faktor Cloud ist immer gegeben, sobald der Nutzer eine Internetverbindung zur Verfügung hat. Außerdem kann durch die Netzwerkstruktur der Zugriff von nahezu jedem Gerät, welches vom Cloud Connector unterstützt wird, stattfinden. Dieses Produkt wird demnach von Atos als Drittanbieter vertrieben. Es wird sich sowohl um die Einbindung als auch den Erhalt nach Launch, in From von SLAs, angeboten. [1]

Allgemein ist das Netzwerk so aufgebaut, dass es Virtuelle Apps und Desktop-Services gibt, die einem Nutzer zur Verfügung gestellt werden sollen. Diese Apps und Services werden von einem Ressourcenblock, der je nach Bedarf extern von Drittanbietern oder auch vom Anbieter selbst direkt bezogen werden kann, bereitgestellt. Der Cloud Connector bietet dabei die Schnittstelle zwischen Nutzer und Ressourcen. Er entnimmt also die benötigten Ressourcen und stellt diese dem Nutzer auf seinen Endgeräten zur Verfügung.[2]

Funktionen des Cloud Connectors sind dabei wie bereits genannt Apps und Virtuelle Maschinen, aber auch die Verwaltung und Verwendung des Active Directories, Store Front und Endpoint Management. Er kann also sehr flexibel genutzt werden, um einzelne Funktionen in die Cloud zu verlagern.

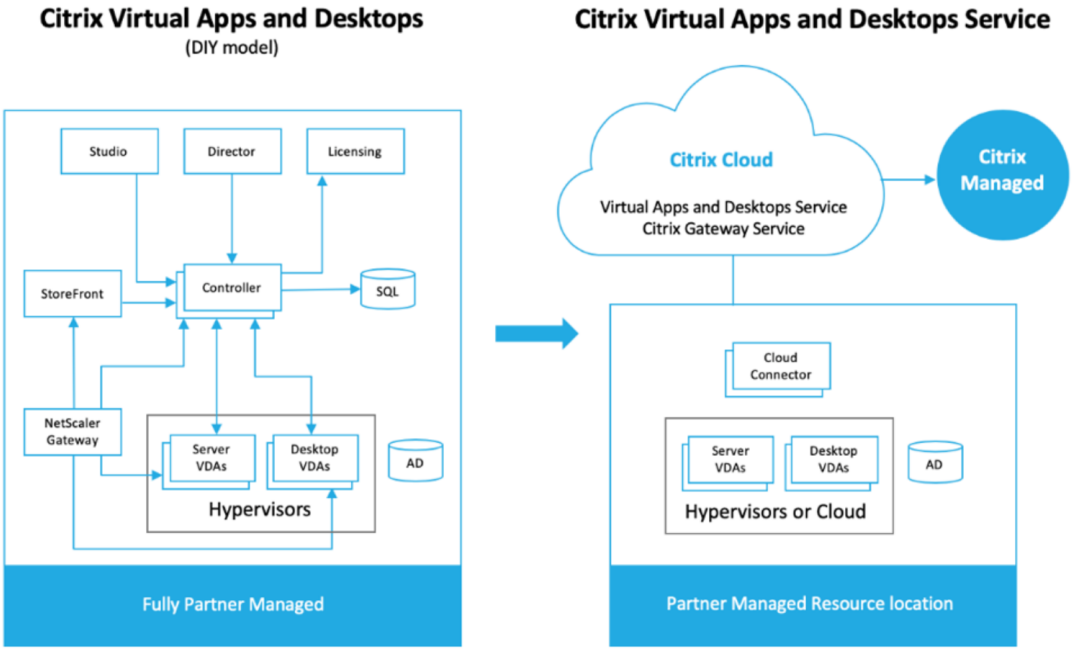

Figure 1: Citrix Cloud Connector Möglichkeiten

Der Grafik zu entnehmen sind hierbei die Unterschiede zwischen einer komplett Bereitstellung des Produkts links und rechts das Modell mit eigenen Ressourcen. Letzteres ist das Modell welches von der Abteilung genutzt wird.

2.3.2 Store Front

Store Front gehört zu den Citrix Produkten und arbeitet in Verbindung mit dem Citrix Cloud Connector. Dieses Produkt ist ein Service, der die Verwaltung von virtuellen Desktops und virtuellen Workstations ermöglicht. Es bietet Single-Sign-On-Zugriff (SSO) auf Anwendungen an und Desktops auf Basis des Citrix NetScaler Gateways. Dementsprechend enthält es weitere Sicherheitsmerkmale wie die Smartcard-Authentifizierung. Im Software Development Kit (SDK)

wird die Anpassung von Benutzeroberflächen und dem App-Deployment durchgeführt. Als Beispiel dafür wäre das Laden von geschäftskritischen Anwendungen nach der Anmeldung zu nennen. [3]

2.3.3 PowerShell

Zur Umsetzung des Skripts wurde das plattformübergreifende Framework PowerShell verwendet. Es ist eine moderne Befehlsshell, die aus verschiedenen Vorteilen anderer Shells entwickelt wurde. Ein Vorteil von PowerShell ist das Akzeptieren und Zurückgeben von .NET-Objekten. [4]

In Bezug auf Skriptsprachen wird sie gerne als Standard zur Konfiguration und automatisierten Verwaltung von jeglichen Systemen genutzt. Dadurch ist sie ideal für die Aufgabe der automatisierten Installationskonfigurationen geeignet.

Für die spätere praktische Anwendung sei zu beachten, dass der Nutzer des Skripts auf dem Zielsystem Administrationsrechte benötigt, um im Vorhinein jeglichen Berechtigungsproblemen präventiv aus dem Weg zu gehen. Zudem sollte das System unter einem Betriebssystem der Firma Microsoft laufen. [5] Sind diese Kernaspekte eingehalten kann das Ausführen von Skripten gewährleistet werden.

2.3.4 PowerShell ISE

Die ISE (Integrated Scripting Environment) ist eine Hostanwendung für PowerShell. Ausgeführt wird diese Anwendung durch den cmdlet ise. Sie bietet eine grafische Oberfläche und dient der Skript Bearbeitung. Durch beispielsweise Syntaxhighlighting sorgt die ISE für eine übersichtlichere Einsicht in das Skript. Funktional bietet sie einen Debugger, Skriptvervollständigung und eine selektive Skriptausführung. [6]

Diese funktionalen Vorteile können jedoch durch Erweiterungen noch weiter

verbessert werden. Als nächstes galt es also eine Erweiterung, mit dem Schwerpunkt Debugger, zu finden. Diese Erweiterung wird im folgenden Kapitel erläutert.

2.3.4.1 ISE Steroids

Bei der Recherche zu einer passenden Erweiterung im Aspekt Debugger kam Power GUI und ISE Steroids am häufigsten auf. Leider wurde ersteres nicht mehr aktiv unterstützt.

Um das Erstellen des Skripts weitestgehend zu vereinfachen, wurde somit auf das Plugin ISESteroids zurückgegriffen. Diese Erweiterung kann durch einen cmdlet, in PowerShell, heruntergeladen und installiert werden. Sobald die ISE gestartet wurde, kann über den Befehl Start-Steroids das Plugin ausgeführt werden. Dies erweitert die ISE um viele verschiedene Funktionen und Einstellungsmöglichkeiten. Beispielsweise können Funktionen einfacher erzeugt, Skriptteile können durch automatische Vervollständigung gebildet und vorgefertigte Funktionen können direkt erzeugt werden. Die jedoch wichtigste Funktion, die für eine gute und schnelle Implementierung geboten wird, ist der vollwertige Debugger. Mit ihm kann das Skript an gewissen Stellen angehalten und Variablenwerte überprüft werden. Einerseits wird dadurch ein klareres Verständnis über das Skript ermöglicht und andererseits lassen sich logische Zusammenhänge schneller Nachvollziehen. [7]

2.3.5 Vorbereitungen

Zur besseren Vorbereitung, der späteren Implementierung, wurde sich mit einem Entwickler aus der Abteilung vernetzt. Dieser zeigte mir einige praktisch bereits umgesetzte Projekte und händigte Skript Beispiele aus, anhand derer sich orientiert, sowie weitergebildet werden konnte. Eines der vorgegebenen Skriptteile konnte spezifisch für die eigene Aufgabe mit einbezogen werden. Die Inhalte des vorgegebenen Skripts waren allgemeine Installation und Download cmdlets

des Cloud Connectors.

Nach dem Austausch setze ich mich mit meinem Betreuer in Verbindung, um die Implementierung und weiteres Vorgehen zu planen. Wie bereits gesagt lag das eigentliche Installationsskript bereits vor, daran mussten nur kleine spezifische Optimierung für die Unternehmensumgebung vorgenommen werden. Diese Aufgabe wurde jedoch zurückgestellt, da sie zunächst nicht elementar war.

Es wurde gemeinsam eine Excel Tabelle mit den zu erledigenden Aufgaben erstellt. Zu den einzelnen Aufgaben sollten klare Termine festgelegt werden. Um diese zu eindeutigen Zeitpunkten erledigt zu haben. Dabei ging es zunächst nicht um die endgültige Fertigstellung und Abgabe der Aufgabe, sondern viel mehr um das gewähren einer besseren Übersicht und klareren Strukturierung des weiteren Vorgehen. Bezogen auf die Ebene des Projektmanagements kann man dabei von festgelegten Sprints reden. Ebenfalls wurde sich zu Daily Huddles verabredet, um sich stetig mit dem Betreuer auszutauschen.

Die Tabelle sieht wie folgt aus:

Aufgabe	Wer	Status	Zieldatum
Erstellung Generalisierung Script zur Übergabe von Parametern.	Löhr	[onGoing]	<24.01>
Erstellung / Überarbeitung Script Basisinstallation Cloud Connectoren	Löhr	[offen]	<04.02>
Erstellung Script Cloud Connector Preferences Check	Löhr	[offen]	<18.02>
Erstellung Script Storefront Preferences Check	Löhr	[offen]	<18.02>
Erstellung Script Storefront Basis Installation	Löhr	[offen]	<25.02>
Erstellung Script Storefront Konfiguration	Löhr	[offen]	<25.02>
Erarbeitung Prozessablauf Diagramm	Löhr	[onGoing]	
Erarbeitung Dokumentation	Löhr	[onGoing]	

2.4 Logischer Aufbau

Wie der Tabelle zu entnehmen ist wurde sich zuerst mit der Übergabe der Installationsparameter beschäftigt. Diese Aufgabe beanspruchte in der Praxisphase am meisten Zeit. Die einzige Vorgabe hierfür war es eine Datei mit den Installationsparametern zu erstellen. Diese Datei sollte entweder das Format XML oder JSON haben. Die Parameterdatei darf später für den Endnutzer die einzige zu editierende Datei sein. Ausgelesen werden diese Parameter aus der Datei in dem Hauptskript, welches die Installation dann durchführt. Je nach Art der Installation gibt es zudem einen Source Ordner mit den Installationsdateien. Im Idealfall soll zudem noch eine log-Datei erstellt werden um den gesamten Prozess zu Dokumentieren. Des Überblicks halber wurde ein Technisches Ablauf Diagramm erstellt.

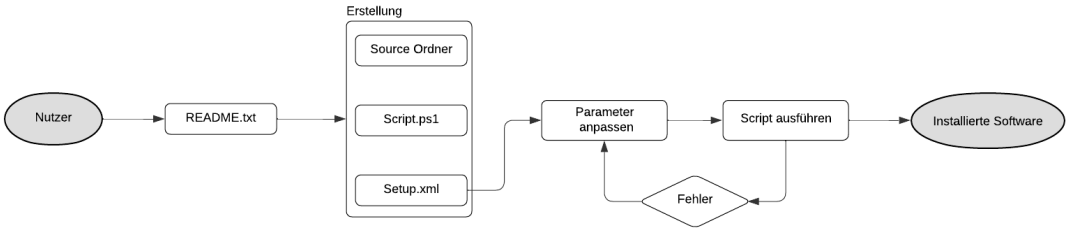

Figure 2: Technisches Ablauf Diagramm

Dieses Ablaufdiagramm soll final eine maximale Übersicht des implementierten Projekts ermöglichen und dem Nutzer dazu dienen die später gegebene Ordnerstruktur besser zu verstehen. Es ist im Verlauf des Projekts stetig ergänzt und angepasst worden.

Zum besseren Verständnis wurden ebenfalls eine ReadMe Datei hinterlegt. Ergänzend ist noch zu Sagen, dass dem Technischen Diagramm die Idee des Parameter Eintragens visualisiert wurde. Fehler die der Nutzer beim Eintragen der Parameter in die Datei macht sollen abgefangen werden und zum korrigieren der Parameter forcieren.

2.5 Automatisierung Citrix Cloud Connector

Dieser Abschnitt thematisiert die gesamte Implementierung des Skripts. Durchweg ist der Aufbau der einzelnen folgenden Abschnitte mit der angeführten Idee und Erklärung, warum sich genau für diese Art der Umsetzung entschieden wurde, strukturiert.

Nachdem ein grundlegendes Verständnis mit Powershell aufgebaut wurde, ging dies über in die Auswahl des Dateityps der Parameterdatei. Festgelegt wurde sich auf das Format XML. Durch die hierarchische Strukturierung der Daten in einer XML kann eine klare Übersicht geschaffen werden. Zudem lassen sich Daten zugänglich und individuell, durch die simple Markup Language, bearbeiten. [8]

2.5.1 Setup.xml

Die Bezeichnung Setup.xml wurde gewählt, um dem Nutzer zu vermitteln, dass dies der Ort zur Anpassung der wesentlichen Parameter für die Installation ist. Der Aufbau der ersten XML Version wurde zuerst wie folgt gewählt:

```
<Setup>
    <CloudConnector>
    <CTXCloudCustomerID>AtosInforma2</CTXCloudCustomerID>
    <CTXCloudClientId>afbe8291-581f-4ff8-z465-e260e062f32e</CTXCloudClientId>
    <CTXCloudClientSecret>Si6XTSBsCJAQbQjweQlyYKyQFuw==</CTXCloudClientSecret>
    <CTXCloudResourceID>e749bfbc-7777-4bd4-8712-fe318ba90974</CTXCloudResourceID>
    <Vendor>Citrix</Vendor>
    <Product>Cloud Connector</Product>
    <PackageName>cwcconnector</PackageName>
    <InstallerType>exe</InstallerType>
    </CloudConnector>
</Setup>
```

Setup bildet den Wurzelknoten, auf den die einzelnen Installationsverzeichnisse folgen. Über diesen Knoten soll später größer skaliert werden (Install, Config, Setup, Remove). Geht man nun in das Element CloudConnector lassen sich acht Attribute finden. In diesen Attributen werden die Werte / Parameter, welche individuell für jede Installation angepasst werden müssen, hinterlegt.

Bei einer größeren Skalierung muss lediglich ein neuer Hüllen Elternknoten erstellt werden.

Dieses Schema langte zunächst und bestätigte sich als praktikabel für die Tests, bei der Parameterübergabe an das CCC Skript.

Während dem Testen wurde eine dynamische Anpassung für die Dateien in der XML integriert. Diese Dateien waren die Store Front Installations exe und ein Backup mit dem Ablageort (URL).

```
<Setup>
        <PackageName>CitrixStoreFront-x64.exe</PackageName>
        <ConfigZip>backup</ConfigZip>
        <HostBaseURL>https://serverB.example.com</HostBaseURL>

        <CloudConnector>
            <CTXCloudCustomerID>AtosInforma2</CTXCloudCustomerID>

        ...
</Setup>
```

Im späteren Verlauf, als sich Gedanken mit der Skalierung gemacht werden konnte, ist man auf zwei Probleme gestoßen. Zum Einen kam auf, dass das Parsen mit Powershell sich als etwas komplizierter für das anfangs gewählte Schema darstellte. Die Problematik hierbei war, dass das Ausgeben der Knotennamen wesentlich umständlicher war, als das der Werte selbst, die in dem Knoten hinterlegt waren.

Zum Anderen kam die Idee auf, das Skript später so zu generalisieren, dass aus einer Setup.xml nicht nur bereits genannte einzelne Konfigurationen hinterlegt werden sollen, sondern auch Installationen mit Backups und ganze Powershell Skripte.

So wurde sich darauf geeinigt, dass mit generalisierten Blöcken gearbeitet werden soll. Dabei bestehen die einzelnen Blöcke aus eine der zwei Kategorien **< Script >** oder **< Parameter >**. In dem jeweiligen Parameter Knoten ist dann als erstes die Art **< Type >** hinterlegt. Hierbei kann zwischen den Bereits vorher erläuterten Kategorien (Install, Config, Setup, Remove) gewählt werden. Der

Weitere Aufbau besteht dann aus dem bereits bekannten Schema.

Die XML wurde also final zusammengeführt und wie folgt angepasst:

```
<Parameter>
    <Type>Install/Config</Type>

    <ConfigZip>backup</ConfigZip>
    <HostBaseURL>https://serverB.example.com</HostBaseURL>

    <PackageName>CitrixStoreFront-x64.exe</PackageName>
    <PackageParameter>-silent -q -quiet</PackageParameter>

    <CTXCloudCustomerID>AtosInforma2</CTXCloudCustomerID>

    ...
</Parameter>
```

Die Idee mit dem Integrieren von verschiedenen Arten von Skripten in dieser XML ist somit umsetzbar, jedoch langte leider die Zeit nicht eine Routine zu schreiben, um ein solches aus der XML in Powershell zu laden.

2.5.2 Main Skript

Im Main Skript vorhanden sind Funktionen zur XML-Zerlegung, dem Downloaden der Installationsdatei und Anweisungen zur Installation im Grundsätzlichen. Der genaue Aufbau wird demnach in den folgenden drei Unterkapiteln erläutert.

2.5.2.1 XML-Parser Zuvor wurden in der Setup.xml alle Attribute mit Werten festgelegt. Die Aufgabe des XML-Parsers ist es nun, sowohl die Attributnamen, als auch deren Werte auszulesen. Der Fokus lag also zuerst auf dem Auslesen der XML-Daten im Allgemeinen. Gelöst wurde dies durch eine foreach-Schleife, die über XML-Pfadbeschreibungen durch die gesamte XML iteriert und die gesammelten Knoten in einem Array speichert. Dieses Array wird dann über eine for-Schleife ausgelesen, um die einzelnen Werte der Knoten in einer

neuen Liste zu speichern. Diese Liste wird dann später bei der Installation weiterverwendet.

Nach der Kernimplementierung wurde sich damit beschäftigt das Skript zu optimieren. So wurde sich um die Fehlerbehandlung und Datenprüfung gekümmert. Einen genaueren Einblick hierfür kann sich im Kapitel Datenprüfung und Fehlerbehandlung eingeholt werden. Dasselbe gilt für das Schreiben und Erzeugen der log-Datei. Fortgefahren wurde dann damit, dass Skriptteile, die eine spezifische Teilaufgabe verfolgen, in einzelnen Funktionen gebündelt wurden. Dies soll die Erstellung von späteren (anderen) Installationen erleichtern, da man gewisse Funktionen universell anwenden kann.

Begonnen wird mit der Funktion LesenderEingabe{}. Zuerst wird die einzulesende XML-Datei ausgewählt. Abgespeichert muss diese dafür in dem aktuellen Dateiverzeichnis sein. Also jenes in dem der Nutzer das Skript abspeichert und ausführt. Sofern eingehalten wird der gesamte XML-Inhalt in der Variable $xmlFile gespeichert. Über diese Variable wird dann mit Hilfe einer Pfadbeschreibung auf den Installationsknoten verwiesen. Der Knoten beinhaltet den spezifischen Input mit den Attributwerten, die für die Installation wichtig sind. Die ausgelesenen Attribute werden mit ihren Werten als Liste dann in $xmlElements abgespeichert und von der Funktion ausgegeben.

Als nächstes werden von der Funktion Get-Nodes{} die in $xmlElements gelisteten Attribute Ausgelesen und in dem Array $returnElemente abgespeichert. Dies dient dazu um in der darauf folgenden Funktion Ausgebenvon-Nullwerten{} die einzelnen Attribute dahingehend zu überprüfen ob diese mit Werten festgelegt sind. Ist dies nicht der Fall so bricht das Skript ab und fordert den Nutzer dazu auf die Attribute mit fehlenden Werten zu belegen.

Sind alle Attribute mit Werten belegt geht es in die letzte Funktion des Parsers Get-NodeData{}. Get-NodeData{} ist von der Struktur ähnlich zu Get-Nodes{}. Es werden hier jedoch nun die einzelnen Werte der Attribute in $returnValue in

Form von einer Liste #text abgespeichert. Dabei ist zu beachten, dass die reinen Werte nur durch den Befehl .'#text' aus dem Array gefiltert werden. So können durch den einfachen Aufruf von Get-Nodes{} überall im Skript die Werte der Attribute genutzt werden.

2.5.2.2 Start-Main

Der zentrale Teil des Skripts ist ein individuell anzupassende Teil. Dieser besteht lediglich aus der Start-Main Funktion. Jener Abschnitt muss je nach Art der Software, die Installiert werden soll, für die Software angepasst werden. Entnommen werden können die vorgefertigten Blöcke der einzelnen Installationen aus der Start Datei.

Über den Abschnitt Start-Main wird das Skript im Ganzen ausgeführt. Der Aufbau basiert auf mehreren if-Anweisungen, bei denen jeweils überprüft wird, ob jede Funktion des XML-Parsers Werte aufweist. Sind die Abfragen durchweg erfüllt, werden die zur Installation relevanten Variablen benannt und mit den geparsten Werten belegt.

2.5.2.3 Download / Installation

Sowohl der Abschnitt Download als auch der darauffolgende Teil der Installation (im Skript) beinhalten Code, welcher bereits existierte und nur minimal zur Zusammenführung mit dem Parser angepasst werden musste. Dieser Skriptblock muss je nach Software, die heruntergeladen werden soll mit der Installation angepasst werden. Durch ihn soll dafür gesorgt werden, dass die Software heruntergeladen wird und die Installation stattfinden kann.

Elementar beginnt die Downloadsektion mit der Deklarierung der Variable $DownloadLocCloudConnector. Sie enthält den spezifischen Downloadlink mit bezogenen Daten aus der XML. Darauf wird $TargetLocCloudConnector mit dem Verzeichnis, in dem die Installations .exe geladen werden soll, erzeugt. Überprüft wird dann, ob die Webanfrage erfolgreich ist. Ist dies der Fall, so

wird der Download gestartet. Andernfalls wird der Download übersprungen.

Abschließend wird dann die Installation mit der Deklarierung von $Arguments eingeleitet. Dazu wird der Start-Process cmdlet von PowerShell verwendet. Ist der Code 0 war die Installation erfolgreich. Ansonsten wird der Nutzer durch eine Terminalausgabe über die fehlerhafte Installation und den Statuscode hingewiesen.

2.5.2.4 URL Check Der in der Download Sektion definierte Downloadlink soll zuvor überprüft werden. Bei der Überprüfung soll die Erreichbarkeit und Dauer zurückgegeben werden. Es handelt sich also im Prinzip um einen klassischen Ping. Implementiert werden sollte dieser Check um sicher zu gehen, dass sich das Skript im Falle eines nicht erreichbaren oder fehlerhaften Links nicht aufhängt, sondern dem Nutzer eine Ausgabe liefert.

Umgesetzt wurde diese Ping Routine mit einer do until-Schleife. Diese Schleife wird solange ausgeführt, bis der Ping erfolgreich ist oder nach 5 Sekunden bei keiner Antwort abgebrochen wird.

Um eine klarere Aussage über die Erreichbarkeit der Website zu erhalten wird zudem noch ein Statuscode abgefragt. Dieser Statuscode sollte immer 200 sein, wenn die Website erreichbar ist. Anhand dieses Statuscodes soll dann später der Download gestartet oder abgebrochen werden.

Die Funktion des URL Checkers wurde letztendlich jedoch nicht in das Finale Skript integriert, da sich im Verlauf des Projektes herausstellte, dass davon auszugehen war, dass lediglich ein paar wenige Links genutzt werden und diese generell immer erreichbar seien. Außerdem würde das einbinden dieser Fehlerüberprüfung in das Hauptskript zu einer größeren Unübersichtlichkeit führen und dadurch die Verständlichkeit des Skripts verschlechtern.

Da aber dennoch Arbeitszeit in diese Aufgabe geflossen ist wollte ich dieses Teilskript erwähnt haben. Zu finden ist es ebenfalls im Github-Repository unter

dem Ordner ‘Project’.

2.6 Formale Komponenten

Dieses Kapitel dient der Skriptvollständigkeit und beschäftigt sich mit dem Schreiben einer Log Datei, dem Fehlerabfangen und die Erläuterung der Datenüberprüfung.

2.6.1 Log Datei

Durch die Dokumentierung gewisser Prozesse eines Programmablaufs kann eine Log-Datei erzeugt werden. Dabei sollte darauf geachtet werden, wie sinnvoll die gesammelten Daten sind und welchen Nutzen sie für beispielsweise spätere Analysen bieten.

Diese späteren Analysen können sowohl während der Entwicklung, als auch nach Fertigstellung des Skripts erfolgen. Während des Entstehungsprozesses dienen sie der Identifikation und Rückverfolgung von Fehlern (Debugging). Nach der Fertigstellung wiederum eher dem Monitoring und Nachvollziehen was exakt passiert. [9]

Eine Einbindung einer Schnittstelle, zur zentralen Analyse der Log Datei, wie zum Beispiel bei einem SIEM in der Netzwerk Security, war nicht nötig. Aus dem Grund da es bei einer solch überschaubaren Skriptgröße nicht sonderbar viel Information zu sammeln gibt. Zudem wäre dies aus zeitlichen Gründen und dem Aspekt der Ressourceneffizienz nicht sinnvoll.

Zum Erstellen der Logfile wurde mit dem cmdlet Start-Transcript, der in der PowerShell Bibliothek enthalten ist, gearbeitet. Beim Verwenden der cmdlets muss zunächst ein Verzeichnis angegeben werden, in dem die log-Datei gespeichert werden soll. Bei Beendigung des Skripts sollte das Schreiben der log-Datei abgeschlossen werden. Mit Hilfe von unterschiedlichen Parametern kön-

nen bestimmte Ereignisse spezifisch in die Datei mit eingetragen werden.[10]

2.6.2 Datenprüfung / Fehlerbehandlung

Als erstes wurde der Dateipfad auf Lese- und Schreibrechte geprüft. Umgesetzt wurde dies mit Hilfe von try-catch Blöcken. Es wird versucht (try) das aktuelle Verzeichnis, in dem das Skript gestartet wurde zu lesen und zum Anderen die log-Datei in jenes Verzeichnis zu schreiben. Ist dies nicht möglich (catch), wird der Nutzer über das Terminal auf fehlende Rechte hingewiesen. Weitere try-catch Blöcke wurden beim Abrufen der XML-Datei und Daten verwendet.

Ist der Abruf der Datei oder der Werte in der XML nicht möglich, wird dies ebenfalls über das Terminal mitgeteilt. Zudem wird bei einer fehlerhaften XML-Dateiabfrage dem Nutzer der spezifische Exception Code ausgegeben. Dies wurde lediglich an dieser Stelle im Skript ausprobiert. Da jedoch davon auszugehen ist, das der Exitcode nicht für jeden Nutzer verständlich ist, wurde im Gesamten mit den bereits genannten Terminal Text ausgaben, die über den cmdlet Write-Host ausgegeben werden, gearbeitet.

Die Überprüfung auf Nullwerte, der entnommenen Daten, findet in der Funktion AusgabeVonNullwerten{} statt. Umgesetzt wird dies mit einer if-Abfrage auf Nullwerte der Daten, welche in Form eines Arrays abgespeichert wurden.

2.6.3 README.txt

In der Readme.txt werden die Installationsanleitungen zu den einzelnen Schritten des Skripts ausgegeben. Der Nutzer soll sich damit durch den gesamten Prozess durcharbeiten können. Orientiert wurde sich beim Erstellen der Readme.txt an die Anleitungen aus der Dokumentation von Great Learning.

Dieses Skript wird dazu verwendet die Installation des CitrixCloudConnec-

tors weitgehend zu automatisieren.

Dafür müssen einzelne Parameter zur Installation konfiguriert werden.

Vor der Ausführung: - Ordner Source anlegen (enthält Installations exe) - Setup.xml nach Schema erstellen - Parameter mit personalisierten Werten festlegen

```
<Setup>
    <CloudConnector>
    <CTXCloudCustomerID>AtosInforma2</CTXCloudCustomerID>
    <CTXCloudClientId>afbe8291-581f-4ff8-z465-e260e062f32e</CTXCloudClientId>
    <CTXCloudClientSecret>Si6XTSBsCJAQbQjweQlyYKyQFuw==</CTXCloudClientSecret>
    <CTXCloudResourceID>e749bfbc-7777-4bd4-8712-fe318ba90974</CTXCloudResourceID>
    <Vendor>Citrix</Vendor>
    <Product>Cloud Connector</Product>
    <PackageName>cwcconnector</PackageName>
    <InstallerType>exe</InstallerType>
    </CloudConnector>
</Setup>
```

Datei und Ordner Bezeichnung darf nicht abweichen!

2.6.4 Prozessdiagramm

Zur Vorstellung eines Kunden und ebenfalls dem besseren Verständnis des Nutzers wurde zudem noch ein Prozessdiagramm angefertigt. In dem folgenden Prozessdiagramm kann der Ablauf des gesamten Arbeitsprozesses abgearbeitet werden. Unterteilt wird der Ablauf hier in drei Abschnitte System, Installation und Software. Es folgen lediglich auf das System und die Installation Aktionen und Ereignisse, da die Software als finaler Abschnitt zu betrachten ist.

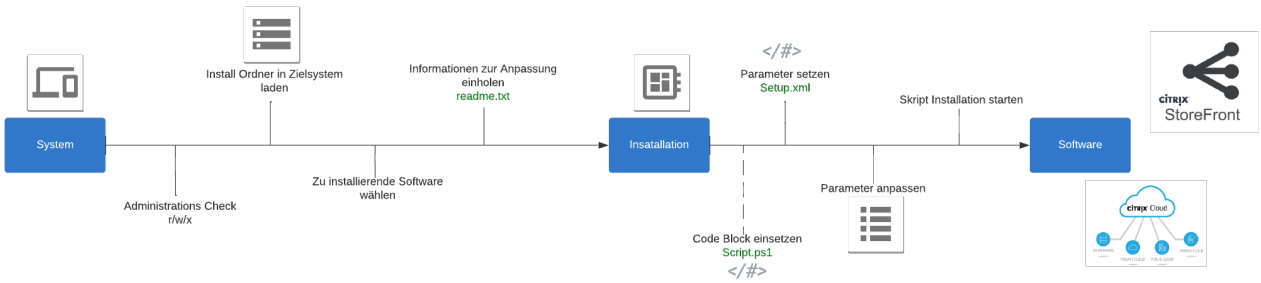

Figure 3: Prozessdiagramm

Beginnt man nun bei dem System, so ist der erste Schritt die Überprüfung der Administrationsrechte. Der Nutzer des Skripts sollte schauen, dass er Lese-, Schreib- und Ausführrechte auf dem System hat. Diese administrativen Rechte sind notwendig, damit der Nutzer jegliche, für die Installation notwendigen, Aktionen auf dem System durchführen kann.

Nach diesem Check kann der heruntergeladene Installationsordner in das Zielverzeichnis geladen werden. Im nächsten Schritt wird die zu Installierende Software gewählt und sich zum weiteren Vorgehen über die readme.txt informiert.

Dann kommt es zu dem eigentlichen Teil der Installation. Dort wird zuerst der Codeblock, der für die Installation notwendig ist, ausgewählt und in das Script.ps1 eingesetzt. Nun können die Parameter, die für die Installation wichtig sind in der Setup.xml gesetzt und angepasst werden. Final wird dann das Skript ausgeführt und die Installation automatisch durchgeführt. Als finales Produkt erzeugt ist dann die gewünschte Software installiert.

2.6.5 Kommentare

Die klassische Herangehensweise beim Programmieren ist es Kommentare passend zu implementierten Code zu verfassen. Der Vorteil darin liegt augenscheinlich darin, dass der Code schneller verständlich ist und man sich auch noch nach einer längeren Zeit einfach in den Code wieder einarbeiten kann. Dabei werden Zeilen oder Funktionen an Code einzeln kommentiert.

Von Powershell aus gibt es die sogenannte kommentarbasierte Hilfe. Kommentiert man seine Funktionen nach diesem Schema, so ermöglicht dies eine praktikable Erweiterung. Man kann sich die Möglichkeiten der Funktion, sowie die Bedienung, in Bezug auf die Parameterübergabe, ganz einfach mit dem Get-Help Command anzeigen lassen [11].

```
Get-Help inputXML -full
```

Diese Erweiterung galt es testweise zu implementieren und auszuprobieren:

```
function inputXML
{
  <#
      .SYNOPSIS
      Setup.xml gets read
      .DESCRIPTION
      Setup.xml gets read, has to be in the current folder from which
      the script is executed
      .INPUTS
      No Inputs needed file Path static
      .OUTPUTS
      All elements which are read from the selected node are outputted.
      .NOTES
      The selected data path has to be adjusted by the prefered data.
  #>

...
```

2.6.6 GUI

Um dem Nutzer den Gebrauch der Software noch einfacher zu machen gab es zu diesem Projekt zuletzt noch die Aufgabe sich an einem grafischen Benutzer Interface zu versuchen. Hierzu wurden die Kenntnisse aus der Ersten Praxisphase und dem Programmieren Modul zwei angewendet. Dies Bot sich an, da in beiden Projekten bereits mit der Erstellung einer GUI gearbeitet wurde.

Angefangen wurde mit der Einbindung spezifischer .NET Bibliotheken zur Erzeugung von grafischen Elementen. Der Grundaufbau ist der folgenden Grafik zu entnehmen:

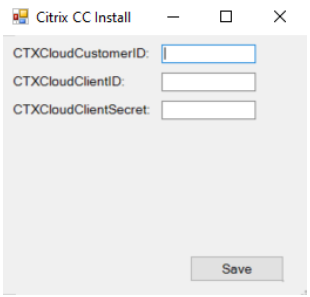

Figure 4: GUI Aufbau

Es wurde eine Main Form erstellt, welche die Hauptseite des Programms

darstellt. Auf der Form wurden Label zur Kennzeichnung der einzelnen Parameter eingetragen. Hinter die Label wurden die Textboxen gesetzt, in welchen der Nutzer die Werte für die einzelnen Parameter eingeben kann. Die Textboxen wurden mit einem Button *Save* verbunden, welcher die Eingabe überprüft und in einem Array *$data* zwischenspeichert.

Nachdem dieses Kernkonzept ausgearbeitet und funktionsfähig war, gab es noch die folgende Funktionen zu erledigen. Je nach Art der Konfiguration sollen die Parameter Label dynamisch angepasst werden. Zudem soll ebenfalls die Menge der Angezeigten Label und Textboxen dynamisch sein. Zuletzt muss noch die Übergabe der gesammelten Werte in die Setup.xml Datei stattfinden.

Als Erstes wurde sich an das dynamische erzeugen der einzelnen Label und Textboxen gesetzt, da dies viel redundanten Code mit sich brachte, demnach aus Effizienzgründen gekürzt werden konnte. Die Idee bestand darin, dass die Parameternamen in einem Array gespeichert werden. Dieses Array soll dann mit Hilfe einer for-Schleife durchlaufen werden und die entsprechenden Label und Textboxen angezeigt werden. Die Schleife sah wie folgt aus:

```
    $data = @("CTXCloudCustomerID", "CTXCloudClientId", "CTXCloudClientSecret",
     "CTXCloudResourceID", "Vendor", "Product", "PackageName", "InstallerType")

    for($i=0; $i -le $data.Length; $i++){
        $current = $data[$i]
        $value = New-Object System.Windows.Forms.Label
        $value.Text = $current
        $y_Label = 10
        $value.Location = New-Object System.Drawing.Point(10,$y_Label)
        $value.AutoSize = $true
        $y_Label+30 | out-null
        $main_form.Controls.Add($value)
        $value = 0 | out-null
}
```

Da diese GUI nur eine Erweiterung zu dem bereits fertigen Skript war, wurde nach dieser Schleifen Fertigstellung nicht weiter an der GUI gearbeitet. Die erwähnten noch zu implementierenden Funktionen blieben also aus. Das Skript ist unter *Project/GUI* zu finden.

Der Grund für das Abrupte abbrechen dieser Aufgabe, lag darin, da akut Hilfe bei bei einem anderen Projekt benötigt wurde. Dieses wird im nächsten Kapitel erläutert.

2.6.7 XML Parsing

2.7 Automatisierung Active Directory

Nach dem sammeln an Erfahrung im Umgang mit Power Shell und der gelungenen Implementierung des Cloud Connector Skripts konnte zu einem neuen Projekt und Anwendungsgebiet gewechselt werden. Dem Active Directory.

2.7.1 Active Directory

Zunächst musste ein Verständnis für das Active Directory (AD) erlangt werden. Dieses ist ein Verzeichnis, in welchem Benutzer auf einer zentralen Datenbank gespeichert sind. Der Vorteil einer zentralen Speicherung ermöglicht die einfachere Verwaltung und Handhabung der Accounts. Will beispielsweise ein Nutzer die Möglichkeit haben auf jedem Rechner ein Benutzerkonto zu haben, so muss nun nicht mehr auf jedem Rechner ein Benutzerkonto erstellt werden, da es zentral Abgerufen wird. Vorteilhaft ist ebenfalls der beispielhafte Fall eines Passwortwechsels. Das Passwort wird lediglich im AD geändert und nicht auf jedem einzelnen System.

Jedem Konto sind durch klare Festlegung spezifische Rechte zugeteilt. Möchte man jenes Konzept größer skalieren, so stößt man schnell auf das Erstellen von Gruppen. Diese Gruppen werden auch Domänen genannt. Eine Domäne gruppiert verschiedene Accountarten, wie Admin-, Berater- oder einfacher Nutzeraccounts. Durch die Richtlinien für unterschiedliche Gruppen lässt sich eine Anpassung deutlich schneller durchführen [12].

Zur besseren Verwaltung wird mit sogenannten GPO (Group Policy Objects) gearbeitet. Diese sind eine Art virtueller Sammlung von Richtlinien Einstellungen. Diese Richtlinien werden in einer Datei gespeichert, die einen eindeutigen Namen zugewiesen bekommt. Diese Datei wird in der Regel in einem Ordner des Computers gespeichert und redundant dazu im AD [13].

Bei der Einarbeitung wurde die Datei Struktur des ADs erläutert und auf die Erzeugung einzelner GPOs eingegangen. Leider kann hierauf nicht detaillierter eingegangen werden, da es sich um Kundenspezifische Daten handelt. Die Aufgabe der zu implementierenden Skripte bezog sich auf die Server Konfiguration mit GPOs.

Einleitend gab es dazu kleine Aufgaben zu lösen, wie das Anlegen verschiedener Organisationseinheiten. Da es sich um einzeilige Kommandos handelte musste lediglich kurze Recherche betrieben werden, um diese zu finden. Das Ziel dieser Aufgabe lag darin, den erhalten Zugang zum AD zu testen.

Aufgaben [14]:

- Script zum Anlegen von Active Directory Organisationseinheiten (Multi)

  ```
  New-ADOrganizationalUnit "test"
  ```

- Script zum Anlegen von Gruppen (Global Groups, Local Groups, Universal Groups)

  ```
  New-ADGroup -Name PS-Test -GroupScope DomainLocal
  ```

2.7.2 GPO

Aufgabe für dieses Kapitel war es, ein Skript zum Handeling mit GPOs zu schreiben. Dieses Skript ist unter *Project/GPO* zu finden. Themen

3 Implementierung von automatischen Standard Service Requests

Um nun die Übergeordneten Prozesse

3.1 Aufgabenstellung

Der Kunde besitzt keine Standard Service Requests. Die Aufträge kommen als Incidents und können demnach nicht automatisiert abgearbeitet werden. Das Reporting ist demnach falsch und die ITIL Konformität nicht gegeben. In Audit ist dies als zu verbessernd identifiziert.

Das Ziel der Arbeit ist es sowohl die vollständige Automatisierung als auch Trennung von Incidents und Standard Service Requests durchzuführen.

3.2 Einordnung der Aufgabenstellung in übergeordnete Prozesse

Es soll die Analyse stattfinden, um dies an die jeweiligen Betriebseinheiten zur Umsetzung weiterzugeben.

Die Analyse setzt sich zusammen aus der Absprache mit den einzelnen Betriebseinheiten, zu aktuellem Vorgehen. Im weiteren Verlauf soll dann das erlangte Wissen an Spezialisten weitergetragen und entwickelt werden. Im Nachgang soll dann mit dem Kunden ein Ende zu Ende Test gemacht werden, um den Erfolg und die vollständige Abdeckung der Arbeit zu prüfen. Der Kunde aktiviert das Formular für den erhaltenen Prozess.

3.3 Praktische Lösung

Die erste Aufgabe war das ganzheitliche Nachvollziehen von Prozessen und Strukturen. Gemacht wurde dies um die Analyse korrekt und zielgeführt durchzuführen. Dafür wurden einige verwendete Frameworks, die bei einem SSR (Standard Service Request) kurz erläutert. Im Folgenden werde ich diese kompakt erläutern und aufzeigen inwiefern jene im Zusammenhang mit dem angebotenen Produkt stehen.

Top Desk Service Now Beatbox –> Microsoft Orchestrator

Im Verlauf der Praxisphase kristallisierte sich immer weiter heraus wie das Vorgehen bei einer Prozessautomatisierung gehandhabt wird. Einleitend

Usecaseanalyse

Danach wurde sich mit der Entwicklerabteilung ausgetauscht in wie weit der Prozess aktuell gehandhabt wird

3.4 Themen die beleuchtet werden könnten

- Wie werden zu automatiseriende Prozesse ausgewählt? (Atos arbeitet mit Konzept alles automatiseren was geht) Kunde legt in SLA von Atos fest was automatisert werden soll

3.5 Microsoft Visio

3.6 Verknüpfung zu Vorlesungsinhalten

Softwareengeneering Userstories

3.7 Kritische, inhaltliche Reflexion von Theorie und Praxis

Links für 4.Praxis

interner Sharepoint https://atos365.sharepoint.com/sites/190118162/Shared%20Documen

4 General Structure

Problemstellung aufführen

Grundlagenaufbau -Literatur mit Ansätzen (Was gibt es schon) -Warum habe ich was wie gemacht (Wenn von Firma vorgegeben trotzdem eventuelle eigene Aspekte alternativ Wege integrieren) -gelerntes im Studium aufbringen

Codehandeling: -in Anhang verlinken -nur interessante/spezifische Aspekte aufzeigen (Segmente des Codes)

Lösung Struktur nach eigenem Ermessen Grundlagen wenn notwendig erklären

5 Ausblick von Automatisierung

6 Quellen

[1] Referenzarchitektur: Virtual Apps and Desktops Service von Citrix Service Provider. Im Internet: https://docs.citrix.com/de-de/tech-zone/design/reference-architectures/csp-cvads.html; Stand: 25.07.2022

[2] Citrix Cloud Connector | Citrix Cloud. Im Internet: https://docs.citrix.com/de-de/citrix-cloud/citrix-cloud-resource-locations/citrix-cloud-connector.html; Stand: 25.07.2022

[3] Citrix StoreFront. Im Internet: https://www.computerweekly.com/de/definition/Citrix-StoreFront; Stand: 26.07.2022

[4] sdwheeler. Was ist PowerShell? - PowerShell. Im Internet: https://docs.microsoft.com/de-de/powershell/scripting/overview; Stand: 21.07.2022

[5] Iwaya A. Does PowerShell Work on Other Operating Systems Besides Windows? Im Internet: https://www.howtogeek.com/306261/does-powershell-work-on-other-operating-systems-besides-windows/; Stand: 21.07.2022

[6] sdwheeler. Einführung in die Windows PowerShell ISE - PowerShell. Im Internet: https://docs.microsoft.com/de-de/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise; Stand: 21.07.2022

[7] . Im Internet: http://www.powertheshell.com/; Stand: 21.07.2022

[8] Extensible Markup Language (XML) 1.0. 1998. Im Internet: https://web.archive.org/web/20060615212726/http://www.w3.org/TR/1998/REC-xml-19980210; Stand: 22.07.2022

[9] online heise. Besser zentral: Professionelles Logging. Im Internet: https://www.heise.de/ratgeber/Besser-zentral-Professionelles-Logging-2532864.html; Stand: 25.07.2022

[10] sdwheeler. Start-Transcript (Microsoft.PowerShell.Host) - PowerShell. Im Internet: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.host/start-transcript; Stand: 22.07.2022

[11] joeyaiello. Beispiele für die kommentarbasierte Hilfe - PowerShell. Im Internet: https://docs.microsoft.com/de-de/powershell/scripting/developer/help/examples-of-comment-based-help; Stand: 29.07.2022

[12] Czernik A. Active Directory und Domäne – einfach erklärt. 2016. Im Internet: https://www.dr-datenschutz.de/active-directory-und-domaene-einfach-erklaert/; Stand: 27.07.2022

[13] REDMOND\\markl. Group Policy Objects. Im Internet: https://docs.microsoft.com/en-us/previous-versions/windows/desktop/policy/group-policy-objects; Stand: 28.07.2022

[14] Verwalten von Organisationseinheiten mit PowerShell. Im Internet: https://blog.netwrix.de/2020/01/24/verwalten-von-organisationseinheiten-und-verschieben-ihrer-objekte-mit-powershell/; Stand: 29.07.2022