Network Unit Testing System

Studienarbeit FS-2020

26. Mai 2020

Autoren:

Mike Schmid
mike.schmid@hsr.ch

Janik Schlatter
janik.schlatter@hsr.ch

Supervisors:

Prof. Stettler Beat
beat.stettler@hsr.ch

Baumann Urs
urs.baumann@hsr.ch

Aufgabenstellung

Änderungen an Netzwerkumgebungen werden in der Praxis auch heute noch durch Kommandozeilenbefehle oder Skripte getestet. Diese Tests beinhalten oft einfache Befehle wie 'ping' oder 'traceroute'. Im Vergleich dazu werden Softwareprojekte durch automatisierte Tests, welche regelmässig ausgeführt werden, getestet. Sogenannte Unit Tests werden vor und nach einer Änderung durchgeführt, um zu Testen, ob sich ein Programm weiterhin innerhalb der geforderten Betriebsparameter verhält. Somit können Fehler schnell gefunden und behoben werden und die Robustheit der Software wird erhöht. Ein vergleichbarer Arbeitsablauf soll auch für den Netzwerkbereich ermöglicht werden.

Eine frühere Studienarbeit hat sich mit der Entwicklung einer Beschreibungssprache befasst, mit derer solche Tests möglich wären. Die Vorarbeit wurde dabei so entwickelt, dass das Programm mit dem Automationsframework SaltStack ausgeführt wurde. Diese Arbeit soll ein Programm entwickeln, welches selbstständig und unabhängig von anderen Programmen arbeiten kann.

Die Studierenden erhalten die Aufgabe, ein Python-Programm zu entwickeln, welches automatisierte Tests auf ein Netzwerk durchführen kann. Das Programm soll gemäss einer Testdefinition selbstständig die auszuführenden Tests erstellen, durchführen und die Testresultate mit einem Erwartungswert vergleichen. Die Auswertung der Tests soll direkt bei der Ausführung auf der Konsole angezeigt werden und zusätzlich in einem Testreport für die spätere Ansicht gespeichert werden. Die Programmausführung kann manuell oder automatisch über einen Deployment-Prozess gestartet werden.

Prof.Beat Stettler

Urs Baumann

Abstract

Die Durchchführung von Tests wird im Netzwerkbereich auch heute noch von Hand durch die Verwendung von Kommandozeilenbefehle wie 'Ping', 'Traceroute' oder diverse Show-Befehle durchgeführt. Wenn bei Konfigurationsänderungen ein Fehler gemacht wird und dieser nicht direkt durch einen Test gefunden wird, kann das dazu führen, dass das gesamte Netzwerk zu einem zufälligen Zeitpunkt einem Ausfall erliegt. In der Softwareentwicklung werden seit einigen Jahren sogenannte Unit-Tests durchgeführt, um die Stabilität der Sowftware zu Testen und zu gewährleisten. Diese Tests werden entweder Regelmässig durch die Entwickler oder automatisiert durch ein Programm durchgeführt, wenn Änderungen am Programm vorgenommen werden. Dadurch lassen sich Fehler früh in der Entwicklung erkennen und Fehler die früh erkannt werden, sind günstiger zu beheben, als Fehler die erst im laufenden Betrieb erkannt werden.

Ziel dieser Studienarbeit ist, eine Software zu entwickeln, mit der man Unittests auf beliebige Netzwerke ausführen kann. Ein starker Fokus wurde dabei auf die Erweiterbarkeit der Software um weitere Tests gelegt. Weiterhin soll darauf geachtet werden, dass die Software möglichst unabhängig von externen Programmen ausgeführt werden kann.

Aus dieser Arbeit ist eine Python-Software entstanden, mit der man automatisiert Netzwerktests auf beliebige Netzwerke ausführen kann. Die Software baut auf dem Nornir-Automatisierungs-Framework auf, ein Python-Modul, welches eine Vielzahl von Methoden und Kommunikationsschnittstellen bietet, mit denen man mit Netzwerkgeräten kommunizieren kann. Die Umsetzung als Reines Python-Programm erlaubt es, die gesamte Software zu erweitern, ohne dass neben Python eine weitere Programmiersprache erlernt werden muss. Neue Netzwerktests lassen sich einfach hinzufügen, ohne dass dabei ein grosser Teil der Software angepasst werden muss.

Management Summary

Ausgangslage

Fehler in Teilbereichen von Netzwerksystemen können dazu führen, dass das ganze System nicht mehr funktioniert. Aus diesem Grund ist es essenziell, dass selbst kleine Änderungen an Netzwerken getestet werden können. Diese Tests werden meistens von Hand oder durch Skripte durchgeführt. Ein Tool, welches das automatisierte Testen von Netzwerksystemen ermöglicht, kann dabei helfen, Fehler zu erkennen, bevor sie zu einem Problem werden.

Vorgehen, Technologien

Zu Beginn wurde eine Domänenanalyse durchgeführt, um die Akteure und Bestandteile einer Netzwerkumgebung zu bestimmen und die zu entwickelnden Netzwerktests zu evaluieren. Darauf aufbauend wurden die funktionalen und nichtfunktionalen Anforderungen an die Software spezifiziert. Auf dieser Basis wurde die Softwarearchitekur ausgearbeitet und mit der Entwicklung begonnen.

Das Programm wurde in der Programmiersprache Python geschrieben und beinhaltet das Modul "Nornir", ein Framework, welches automatisierte Tasks auf Netzwerksysteme, wie z.B. Konfiguration oder Tests, ermöglicht.

Ergebnisse

Aus dieser Studienarbeit ist ein Python-Programm entstanden, welches Netzwerktests, die in einer Definitionssprache spezifiziert werden, gegen ein Netzwerk durchführt und die Ergebnisse selbstständig auswertet und dem Benutzer anzeigt. Nornir erlaubt dabei, eine Vielzahl von Netzwerkgräten anzusprechen, welche über herkömmliche Methoden wie SSH umfänglicher zu testen wären.

Die Software lässt sich ohne Installation auf jedem Gerät ausführen, welches Python-Code ausführen kann, unabhängig vom Betriebssystem. Geräte, auf denen Python nicht installiert ist, müssen dies zuerst Installieren, können das Programm danach aber ohne weiteres ausführen.

Dadurch, dass das Programm reiner Pyhton Code ist, lässt es sich einfach in ein bestehendes Tool für die kontinuierliche Integration einbinden. Die Testdefinitionen lassen sich über ein Versionsverwaltungstool zentralisieren, so dass mehrere Netzwerkleute gleichzeitig Tests für eine Umgebung entwickeln können.

Inhaltsverzeichnis

I. Technischer Bericht

1 Einleitung

1.1 Problemstellung

Netzwerke bestehen aus dutzenden bis tausenden Komponenten. Jede dieser Komponente hat eine eigene Konfiguration und Aufgabe. In den letzten Jahren ist es durch die softwaregesteuerte Konfiguration von Netzwerkgeräten einfacher geworden, die Komponenten für ihre Aufgabe einzustellen. Trotzdem werden die Überprüfungen der Konfiguration auch heute noch manuell vorgenommen. Dies kann dazu führen, dass wegen menschlicher Fehler ein Fehlverhalten eines der Netzwerkkomponente dazu führt, dass das gesamte Netzwerk gestört wird. Weiterhin wird die Überprüfung noch komplizierter, da neben der statischen Konfiguration sich das Netzwerk zur Laufzeit dynamisch anpasst, um die Performanz des Systems zu optimieren und Fehler sowie Ausfälle zu korrigieren. Dieses Verhalten wird über verschiedene Netzwerkprotokolle gesteuert, z.B. OSPF oder BGP.

In der Softwareentwicklung werden schon seit Ende der 80er-Jahre Komponententests, sogenannte Unit-Tests durchgeführt, um einzelne Komponenten (Units) automatisiert zu Testen. Dabei wird ein Computerprogramm ausgeführt, welches mit verschiedenen Eingabeparametern überprüft, ob die Ausgabe des zu testenden Programms den erwarteten Ergebnissen entspricht.

Dabei ist es möglich die Tests vor und nach einer geplanten Änderung durchzuführen, um zu überprüfen, ob die Software innerhalb der definierten Funktionsparameter operiert. Tests sollen dafür so geschrieben werden, dass möglichst jede Situation mit den Eingabeparametern abgebildet wird. Unittests sollen regelmässig durchgeführt werden, damit Fehler früh gefunden werden und sich nicht auf das gesamte System auswirken können.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist, ein Programm zu entwickeln, mit dem sich Netzwerktests mit der gleichen Arbeitsweise durchführen lassen, wie Unittests in der Softwareentwicklung durchgeführt werden. Dabei müssen folgende Anforderungen an die Tests erfüllt sein:

- Tests müssen planbar sein. Es soll ein Testplan existieren.
- Tests müssen systematisch spezifiziert werden. Es existieren Test-Spezifikationen.
- Testresultate werden Dokumentiert.
- Tests sollen, wo möglich, automatisiert durchgeführt werden.
- Testergebnisse müssen reproduzierbar und nachvollziehbar sein.

Es soll evaluiert werden, welche bereits verfügbaren Tools sich für ein solches Testprogramm eignen. Die Umsetzung soll diese Tools einbinden. Eine Wichtige Anforderung an das zu entwickelnde System ist, dass sich weitere Netzwerktests möglichst einfach hinzufügen lassen, ohne dass dafür der gesamte Code geändert werden muss.

1.3 Herausforderungen

Eine der grössten Herausforderungen an ein automatisiertes Testsystem sind die verschiedenen Protokolle und die Unterschiede der Standards von diversen Herstellern.

Ohne Kenntnisse, welche Protokolle auf einem Netzwerkgerät konfiguriert sind, können diese Netzwerkgeräte nicht effizient getestet werden, da die Netzwerkprotokolle das Verhalten der Geräte zur Laufzeit beeinflussen. Deshalb müssen für die Testdurchführung im vornherein die Konfigurationen und verwendeten Protokolle der Netzwerkgeräte bekannt sein und ein Testsystem muss mit diesen interagieren können.

Unterschiedliche Hersteller haben verschiedene Kommandozeilenbefehle (CLI-Commands) für ihre Geräte, welche für die Konfiguration und Abfrage der Konfiguration verwendet werden. Ausserdem kann es vorkommen, dass ein Hersteller mit der Einführung einer neuen Version des Gerätebetriebssystems neue CLI-Commands einführt oder alte Befehle ändert. Dies setzt eine enorme Flexibilität für ein Testprogramm voraus, welches ein beliebiges Netzwerk mit unterschiedlichen Geräten von verschiedenen Herstellern testen soll.

1.4 Vorarbeit

Die Studienarbeit 'Network Unit Testing' aus dem Herbstsemester 2016 von Andreas Stalder und David Meister hat sich bereits mit dem Thema automatisierte Netzwerktests auseinandergesetzt. Der Fokus lag dabei auf dem Ausarbeiten einer Testdefinitionssprache, mit der solche Netzwerktests annotiert werden können. Zudem wurde mit SaltStack ein Konfigurationsmanagement-Tool evaluiert und auf dessen Basis eine Software entwickelt, mit der man Netzwerktests ausführen konnte.

Die grössten Herausforderungen der Vorarbeit lagen dabei auf den Unterschieden verschiedener Hersteller und darin, dass die Outputformate der Neztwerktests je nach Hersteller anders zu Parsen sind. Diese Herausforderungen haben dazu geführt, dass beim Parsen der Ergebnisse oftmals auf reguläre Ausdrücke zurückgegriffen wurde, deren Implementation umständlich und schwierig zu lesen ist. Die Autoren haben dabei die Hoffnung angemerkt, dass man die Resultate künftig einfacher verarbeiten kann und die Hersteller ein einheitliches Rückgabeformat verwenden.

Der grösste Nachteil in der Verwendung der Lösung aus der Vorarbeit besteht darin, dass die Implementation neuer Netzwerktests eher umständlich ist. Ausserdem besteht eine starke Abhängigkeit zum Tool SaltStack, bei dem keine Garantie für eine langfristige Verfügbarkeit besteht.

Zukünftige Arbeiten sollen deshalb eine geringere Abhängigkeit zu externen Tools haben und die Erweiterung um neue Tests so einfach wie möglich umsetzen.

2 Ergebnisse

2.1 Resultat der Arbeit

Die Studienarbeit hat ein Python-Programm entwickelt, welches basierend auf einer Testdefinition im YAML-Format automatische Tests gegen ein Netzwerk ausführen kann.

Kern der Software ist das Nornir-Modul für Python, ein Framework welches in Python geschrieben ist und die Automation von Netzwerktätigkeiten ermöglicht. Die Auswahl von Nornir ist das Ergebnis der Evaluation möglicher Tools für die Software, wobei schon zu Beginn der Analysephase die Verwendung von Nornir feststand, da das Framework ein breites Spektrum von Funktionen für die Durchführung und Auswertung von Netzwerktests bietet. Nornir erlaubt dem Anwender, automatisiert Konfigurationsänderungen oder -abfragen an ein Netzwerk zu senden.

Eine mögliche parallele Ausführung der Netzwerktests wurde in der frühen Phase des Projekts angesprochen, konnte aber aufgrund des engen Zeitrahmens der Arbeit nicht umgesetzt werden. Stattdessen wurde die Zeit investiert, damit die Implementierten Tests korrekt funktionieren und die Erweiterung um neue Tests so einfach wie möglich ist. Die Erweiterung um eine parallele Ausführung wird in die künftigen Arbeiten übernommen.

Die im Kapitel 1.2-Aufgabenstellung genannten Anforderungen an die Tests werden vom Programm vollständig erfüllt:

Planbarkeit von Tests/Test-Spezifikation Die Testdefinition entspricht dem Testplan/ der Test-Spezifikation. Netzwerktests werden in der Testdefinition spezifiziert und in der Defininitionsreihenfolge, sofern nicht über das Grafische Userinterface anders definiert, durchgeführt.

Dokumentation von Testresultaten Die Testresultate werden in einem txt-File mit dem Datum und Zeitpunkt der Durchführung gespeichert. Somit lassen sich sämtliche vergangenen Testdurchführungen nachvollziehen.

Automatische Testdurchführung NUTS2.0 lässt sich vollautomatisch durchführen. Eine Orchestrierungssoftware muss dafür lediglich den Kommandozeilenbefehl 'python -m nuts -r' zu festgelegten Zeitpunkten ausführen und das Programm läuft danach selbstständig durch und dokumentiert die Ergebnisse.

Reproduzierbarkeit/Nachvollziehbarkeit der Ergebnisse Wenn sich die Konfiguration im Netzwerk zwischen der Testdurchführung nicht verändert, wird auch das Testergebnis gleich ausfallen. Testergebnisse werden in einem einheitlichen Format gespeichert und bei nicht bestandenen Tests wird zusätzlich zum Testnamen noch das erwartete und das tatsächliche Ergebnis ausgegeben, um einen Soll-Ist-Vergleich durchführen zu können.

2.2 Programmausführung

Das Programm kann auf einem Beliebigen Betriebssystem, welches Python installiert hat, ausgeführt werden. Mit dem Befehl 'python -m nuts' im Order 'Nuts2.0' wird das Programm gestartet und lädt selbstständig die Informationen aus dem Inventar und die Testdefinitionen. Mit dem Kommandozeilenparameter '-r' kann dabei das Grafische Userinterface übersprungen werden, um Netzwerktests automatisiert ausführen zu lassen. Das GUI wird verwendet, um spezifische Tests zu selektieren und die Reihenfolge der selektierten Tests festzulegen. Wird das GUI übersprungen, werden alle Netzwerktests in der Reihenfolge durchgeführt, in der sie in der Testdefinition angegeben wurden.

Die Testergebnisse werden nach der Testdurchführung in der Konsole angezeigt und in einem Result.txt, unter Angabe des Zeitstempels gespeichert. Erfolgreiche Tests werden nur mit dem Testnamen aus der Testdefinition und dem Vermerk 'PASSED' ausgegeben/gespeichert, da hauptsächlich die nicht bestandenen Netzwerktests relevant sind. Nicht bestandene Tests haben zusätzlich zum Testnamen aus der Testdefinition noch das erwartete Ergebnis und das tatsächliche Ergebnis für einen Soll-Ist-Vergleich.

2.3 Geräte-/Herstellerunterstützung

Das Modul Nornir erlaubt mit seinen Plugins eine breite Unterstützung von diversen Herstellergeräten. Verschiedene Plugins haben dabei ein unterschiedliches Rückgabeformat, welches in den konkreten Netzwerktests in eine Normalform gebracht werden muss, damit NUTS2.0 einheitlich damit umgehen kann.

Nachfolgend sind die verschiedenen Plugins für die Verbindungsschnittstellen beschrieben:

Napalm Abkürzung für "Network Automation and Programmability Abstraction Layer with Multivendor support". Napalm ist eine Pyhton Library welche verschiedene Funktionen anbietet, mit denen man mit Netzwerkgeräten über eine einheitliche Schnittstelle kommunizieren kann.

Paramiko Paramiko ist eine Python-Implementation des SSHv2 Protokolls für die sichere Kommunikation zwischen verschiedenen Endgeräten.

Netmiko Netmiko ist eine Library, welche die Paramiko SSH verbindung vereinfacht. Das Ziel von Netmiko ist, für verschiedene Herstellergeräte eine einheitliche Schnittstelle zu bieten und die Kommunikation zwischen Endgeräten und Server zu vereinfachen.

Netconf Das Network Configuration Protocoll ist ein Protokoll für das Netzwerk Management. Netconf wurde als RFC 6241 publiziert und bietet Mechanismen für die Installation, Manipulation und das Löschen von Konfigurationen auf Netzwerkgeräten.

2.4 Implementierte Netzwerktests

Die nachfolgende Auflistung beschreibt die Netzwerktests, die zum Zeitpunkt des Projektabschlusses implementiert sind. In erster Linie wurden Tests implementiert, die auf das gegebene Testnetzwerk ausgeführt werden konnten. Im Kapitel 3.3-Künftige Arbeiten werden weitere Netzwerktests beschrieben, die in das System integriert werden können.

Testname	Testbeschreibung
Napalm Ping	Test der einen Ping auf ein Zielgerät mit dem Napalm-Treiber durchführt.
Netmiko Ping	Test der einen Ping auf ein Zielgerät mit dem Netmiko-Treiber durchführt.
Napalm show Interfaces	Test der mit dem Napalm-Treiber die Interfaces des Hostgeräts anzeigt.
Netmiko show Interfaces	Test der mit dem Netmiko-Treiber die Interfaces des Hostgeräts anzeigt.
Netmiko Traceroute	Test der die Hops zwischen dem Hostgerät und einem Zielgerät anzeigt.
Napalm show ARP Table	Test der die ARP-Tabelle des Hostgeräts mit dem Napalm-Treiber abfragt.
Netmiko show ARP-Table	Test der die ARP-Tabelle des Hostgeräts mit dem Netmiko-Treiber abfragt.
Napalm OSPF Neighbors	Test der die OSPF Nachbarn des Hostgeräts mit dem Napalm-Treiber abfragt.
Netmiko OSPF Neighbors	Test der die OSPF Nachbarn des Hostgeräts mit dem Netmiko-Treiber abfragt.

Tabelle 1: Implementierte Tests

Man könnte nun argumentieren, dass mehrere Tests doppelt Implementiert seien, was duplizierter Code wäre. Allerdings kann man nicht für beliebige Hostbetriebssysteme die gleichen Netzwerktests verwenden. Zudem gibt es bestimmte Tests, z.B. Traceroute, die sich nicht mit Napalm implementieren lassen (zumindest nicht zum Zeitpunkt der Implementierung, ein Pull-Request für die Integrierung des Napalm Traceroute steht offen). Das Programm entscheidet, basierend auf dem Betriebssystem des Hostgeräts, welcher konkrete Test instanziiert werden soll. Napalm wurde dabei als der default Test für die IOS Geräte des Testnetzwerks implementiert, mit den Netmiko-Tests als Fallback und für Tests, die sich mit dem Napalm-Treiber nicht umsetzen liessen.

3 Schlussfolgerungen

3.1 Endresultat

Das Endergebnis der Studienarbeit hat leichte Abweichungen von dem, was wir zu Beginn des Projekts geplant haben. In der Architektur wurde zuerst eine weitere Strategy-Factory für die Erstellung der Netzwerk-Verbindungen geplant. Diese wurde aber nicht implementiert, da sich herausgestellt hat, dass eine Strategy-Factory nur für weitere Komplexität im Code sorgt. Stattdessen wurde ein Mapping der Hostbetriebssysteme auf die kompatiblen Verbindungsschnittstellen mittels einem Dictionary vorgenommen. Dieses Vorgehen bietet eine ähnliche Funktionalität, ist aber weitaus simpler in der Anwendung und im Verständnis.

Weiterhin wurde ein grosser Teil der Testdurchführungslogik in die konkreten Testimplementationen verschoben. Am Anfang wurde geplant, dass jede Komponente die Logik beinhaltet die für die Testdurchführung, die Evaluation der Ergebnisse und das Mapping der Resultatwerte benötigt werden. Stattdessen wurde die Funktionalität des Mappers, des Evaluators und des Testrunners soweit vereinfacht, dass darin für jeden Test die jeweilige Methode aufgerufen wird, die diese Logik spezifisch für jeden Test implementiert. Durch dieses Vorgehen konnte beispielsweise das Mapping der Testresultate für jeden Netzwerktest innerhalb des Tests vorgenommen werden. Dadurch wurde viel Code im Mapper gespart, da jeder Test eine andere Formatierung der Rückgabewerte hat und dementsprechend für jeden Test ein spezifisches Mapping der Ergebnisse hätte gemacht werden müssen.

Das Nornir-Framework erlaubt es dem Benutzer, schnell und einfach Netzwerkbefehle an ein definiertes Netzwerkgerät zu senden und die Rückgabewerte werden dabei in einem Dictionary von Resultatwerten zurückgegeben. Dadurch müssen die Resultate nicht mit Verwendung von regulären Ausdrücken mühsam geparst werden, sondern sie können unter Anwendung von einfachen Operationen in eine einheitliche Form gebracht und mit den Erwartungswerten verglichen werden. Der grösste Nachteil in der Verwendung von Nornir ist die Navigierung durch die Dokumentation. Will man die einzelnen Netzwerktreiber (Napalm, Netmiko, etc.) verwenden, muss man oftmals in den Dokumentationen der Treiber selbst nachsehen, wie genau man die Implementation nun vornehmen muss, vorausgesetzt, es existiert eine spezifische Anleitung für die Implementation.

3.2 Vergleich mit der Vorarbeit

Die Studienarbeit aus dem Herbstsemester 2016 hat im Vergleich mit dieser Studienarbeit mehr konkrete Netzwerktests umgesetzt. Allerdings hat diese Studienarbeit einen höheren Fokus auf der Erweiterbarkeit um neue Tests und es werden mehr verschiedene Gerätehersteller unterstützt.

Die reine Pyhton-Implementation erlaubt ein Deployment auf eine Vielzahl von Geräten mit unterschiedlichen Betriebssystemen unabhängig von externen Frameworks. NUTS2.0 lässt sich in ein bestehendes Configuration Mangement Tool einbinden und kann auf jedem Gerät ausgeführt werden, welches Python3 unterstützt. Im Vergleich dazu wurde NUTS1.0 hauptsächlich für die Verwendung auf Linux-Systemen entwickelt.

Die Erweiterung des Systems ist in NUTS2.0 einfacher gestaltet, da sämtlicher Code in Python geschrieben ist und die verwendeten Module ebenfalls in Python entwickelt wurden.

3.3 Künftige Arbeiten

Automatische Geräteerkennung

Im aktuellen Programm müssen sämtliche Netzwerkgeräte von Hand in das Inventar aufgenommen werden, was genaue Kenntnisse des Netzwerks voraussetzt und zeitintensiv ist. Eine mögliche Erweiterung von NUTS2.0 wäre eine automatische Erkennung aller Geräte im Netzwerk und das automatische Erstellen des Inventars. Dieses Vorgehen würde dem Anwender einiges an Arbeit einsparen und den Prozess für die Erstellung von Netzwerktests beschleunigen.

Anbindung einer Datenbank

Die Anbindung einer Datenbank würde viele Vorteile mit sich bringen. Die Verwaltung des Inventars oder der Testdefinitionen müsste nicht mehr in YAML-Files gemacht werden, sondern könnte mit einer Datenbank vereinfacht werden. Das Speichern von Testresultaten in einer Datenbank würde es ermöglichen, spezifische Queries für individuelle Testresultate oder Testdurchführungen zu verwenden. Dadurch könnte man wirkungsvolle Abfragen durchführen, um beispielsweise sämtliche Tests vor einer geplanten Änderung und danach abzurufen und auszuwerten.

Komplette GUI geführte Durchführung

Die aktuelle Software hat nur ein minimalistisches GUI für die Auswahl und Reihenfolge der Testdurchführung. Eine mögliche Erweiterung ist das Einbinden eines GUI für die gesamte Testdurchführung. Ein Benutzer könnte z.B. die Testdefinitionen über ein Interface gesteuert erfassen, statt diese in einem YAML selber zu erstellen. Man könnte für sämtliche Testcommands Beschreibungen anzeigen, was diese Tests machen und wie sie verwendet werden. Es könnte eine Grafische Abbildung des Netzwerks angezeigt werden (Digitaler Zwilling), in dem man sämtliche Neztwerkgeräte mit ihren Parametern anzeigt.

Erweiterung um Tests

Zum Zeitpunkt der Studienarbeit wurden nur einige Netzwerktests implementiert, die sich auf das Testnetzwerk auch ausführen liessen. Das System liesse sich um sämtliche Netzwerktests erweitern, die in allen möglichen Netzwerken auch tatsächlich ausgeführt werden könnten. Ausserdem gibt es Netzwerktests, die sich mit Nornir zum jetzigen Zeitpunkt gar nicht ausführen lassen, z.B. die Grafische Darstellung des OSPF spanning Tree. Solche Tests müssten manuell implementiert werden und benötigen allenfalls andere Module als Nornir.

Mögliche Tests Die folgende Auflistung zeigt einige der gängigsten CLI-Befehle, wie sie im Netzwerkumfeld verwendet werden um die Netzwerkumgebung zu testen.

- 'show running-configuration'
- 'show startup-configuration'
- 'show ip protocols'
- 'show ip route'
- 'show interfaces status'
- 'show interfaces trunk'
- 'show interfaces'
- 'show ip ospf interface'
- 'show ip ospf border-routers'
- 'show ip ospf virtual-links'
- 'show ip pgp'
- 'show ip pgp summary'
- 'show ip bgp neighbor'
- 'show ip bgp paths'

Asynchrone Durchführung der Tests

Momentan werden alle Netzwerktests nacheinander, d.H. Synchron durchgeführt. Dadurch benötigt das Programm für die Durchführung von umfangreichen Testdefinitionen viel Zeit. Man könnte in einer zukünftigen Erweiterung Tests mit der gleichen Kategorie, z.B. Ping-Tests asynchron (parallel) ausführen, was zu einer kürzeren Durchführungszeit führen würde.

Automatische Erstellung repetitiver Tests

Eine weitere Möglichkeit für die Erweiterung der Software ist, dass repetitive Netzwerktests automatisch erstellt werden. Beispielsweise könnten Ping tests für sämtliche Geräte, die im Inventory definiert sind vom Programm erstellt und nicht manuell erfasst werden. Weiterhin könnte eine Erweiterung vorgenommen werden, die bestimmte Tests bei der Erfassung vorschlägt, z.B. show ip interface brief für einen Router.

II. Projektdokumentation

1 Projektplanung

1.1 Projektübersicht

Das Testen von Netzwerkkonfigurationen findet auch heute noch hauptsächlich mit handgeschriebenen CLI-Befehlen oder kleinen Skripten statt. Wenn der Netzwerktechniker einen Test vergisst, oder die Formulierung nicht stimmt, kann es vorkommen, dass im Netzwerk Fehler auftreten, deren Ursprung schwierig zu ermitteln ist und eine komplette Repetition der (handgeschriebenen) Tests erfordert.

Ein Programm, welches wie in der Softwareentwicklung vordefinierte und automatisch durchgeführte Tests, sogenannte Unit-Tests, ermöglicht, könnte diese Probleme stark verringern.

1.2 Zweck und Ziel

Die Studienarbeit soll aufzeigen, dass Studierende unter Anwendung der im Studium erlernten Fähigkeiten in der Lage sind, Ingenieurstätigkeiten durchzuführen.

Mit NUTS2.0 soll ein Tool entwickelt werden, welches es Netzwerkleuten erlaubt, Unit Tests in Netzwerkumgebungen durchzuführen. Die Anwendung von Tests erlaubt eine höhere Stabilität und Fehlertoleranz des Systems.

1.3 Projektorganisation

1.3.1 Projektmitglieder

Name	Email
Janik Schlatter	jschlatt@hsr.ch
Mike Schmid	mschmid@hsr.ch

1.3.2 Externe Schnittstellen

Name	Email	Zuständikeit
Beat Stettler	beat.stettler@hsr.ch	Betreuer
Urs Baumann	urs.baumann@hsr.ch	Betreuer

1.4 Managementabläufe

1.4.1 Zeitbudget

Das Projekt wurde am 20.02.2020 gestartet und wird voraussichtlich am 28.05.2020 enden. Das heisst, es stehen 15 Wochen während dem Semester zur Verfügung. Jedes Projektmitglied arbeitet insgesamt 240 Stunden an dem Projekt, sprich 16 Stunden pro Woche pro Projektmitglied. Da der Dienstag 14.04.2020 und der Donnerstag 21.05.2020 jeweils ein Unterrichtsfreier Tag ist, werden die 16 Stunden pro Projektmitglied auf jeweils vier Samstage im Projekt verteilt aufgeteilt. Diese Samstage dienen dem Ziel, die Dokumentation nachzutragen und die Risiken, wie sie im Kapitel 1.5 beschrieben sind, zu minimieren. Dazu gehören Bugfixing, Recherchen und Aufarbeiten von Themen, die ungenügend verstanden sind und Refactoring des Codes.

Projektdauer	15 Wochen
Anzahl Projektmitglieder	2
Arbeitsstunden pro Woche und Person	16
Arbeitsstunden insgesamt	480
Projektstart	20.02.2020
Projektende	28.05.2020

Tabelle 2: Projektparameter

Zeitliche Planung Die 15 Wochen des Projekts werden in fünf Phasen unterteilt: Initialisierung, Analyse, Design, Realisierung und Abschluss.

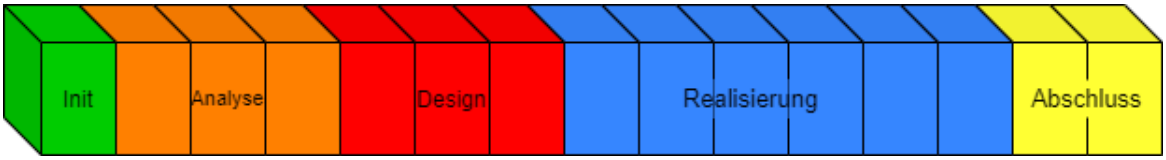

Abbildung 1: Zeitplanung

1.4.2 Projektphasen

Farbe	Bezeichnung	Zeitrahmen
Grün	Initialisierung	1 Woche
Orange	Analyse	3 Wochen
Rot	Design	3 Wochen
Blau	Realisierung	6 Wochen
Gelb	Abschluss	2 Wochen

Tabelle 3: Projektphasen

1.4.3 Meilensteine

Nr	Bezeichnung	Termin	Beschreibung
M1	Projektplan	So 01.03.2020	Grundentwurf der Requirements, Risikoanalyse & -management, Projektorganisation, Managementabläufe, Infrastrukturentwurf, Qualitätsmassnahmen Grundentwurf.
M2	Requirements	So 15.03.2020	Ausgearbeitete Requirements, Nichtfunktionale Anforderung, Zu verwendende Tools und Schnittstellen beschrieben.
M3	Prototyp	So 05.04.2020	Architektur festgelegt, Schnittstellen angelegt, Architekturdokumentation, Erster lauffähiger Prototyp.
M4	Feature Freeze	Do 07.05.2020	Hauptfunktionalität der Software implementiert, Bugs sind bekannt und Dokumentiert, Codedokumentation zu 60% fertiggestellt.
M5	Codefreeze	So 24.05.2020	Bugfixes erstellt, Tests sind alle erfolgreich, Codedokumentation zu 80% fertiggestellt.
M6	Projektabgabe	Do 28.05.2020	Dokumentation fertiggestellt und abgegeben.

Tabelle 4: Meilensteine

1.4.4 Iterationen

Iteration	Inhalt	Start	Ende
Initialisierung	Projektstart und Kick-Off Meeting	20.02.2020	23.02.2020
Analyse 1	Projektplanung	24.02.2020	01.03.2020
Analyse 2	Evaluation Module (Nornir, Napalm, Openconfig)	02.03.2020	08.03.2020
Analyse 3	Requirements & Analyse Testcases	09.03.2020	15.03.2020
Design 1	Architekturdesign	16.03.2020	22.03.2020
Design 2	Testen der Module	23.03.2020	29.03.2020
Design 3	Prototyp programmieren	30.03.2020	05.04.2020
Realisierung 1	Umsetzung Filehandler, Testrunner, Inventarmanagement und Testdefinitionsklassen	06.04.2020	12.04.2020
Realisierung 2	Umsetzung Testerstellung-Strategy-Factory und Testresultat-Mapper	13.04.2020	19.04.2020
Realisierung 3	Umsetzung Connection-Strategy-Factory und Evaluator	20.04.2020	26.04.2020
Realisierung 4	Umsetzung Testbuilder, Testcontroller, und Reporter	27.04.2020	03.05.2020
Realisierung 5	Umsetzung Testreihenfolge, automatische Geräteerfassung und Kommandozeilen-UI	04.05.2020	10.05.2020
Realisierung 6	Fertigstellung und Refactoring	11.05.2020	24.05.2020
Abschluss 1	Bugfixes, Refactoring und Dokumentation	18.05.2020	24.05.2020
Abschluss 2	Fertigstellung Schlussbericht	23.05.2020	28.04.2020

Tabelle 5: Projektiterationen

1.4.5 Arbeitspakete

Die Arbeitspakete sind in vier Kategorien eingeteilt, um eine Übersicht zu bieten.

- Ausführung der Tests - Alles was mit der Ausführung und Evaluation der Netzwerktests zu tun hat.
- Erstellen der Tests - Alle Arbeitspakete, die sich mit der Erstellung der Testsuite auseinandersetzen.
- Hilfsklassen - Filehandler, Logging, Integration Tests und Kommandozeileninteraktion
- Tests - Integration einzelner Tests in das System

Kategorie Ausführung der Tests

Implementation Testrunner

Beschreibung	Die Testrunner-Klasse führt die ihm vom Controller übergebenen Tests gegen das Netzwerk aus und gibt ein Resultat in einem beliebigen Format zurück.
Akzeptanzkriterien	Testrunner ist in der Lage, Tests gegen ein Netzwerk auszuführen und reagiert auf Fehler, indem fehlgeschlagene Tests als solche annotiert werden.
Aufwandschätzung Tatsächlicher Aufwand	6 Stunden 4 Stunden
Priorität	Mittel

Implementation Evaluator

Beschreibung	Der Evaluator vergleicht die Resultate der Netzwerktests mit den aus den Testdefinitionen beschriebenen Soll-Werten.
Akzeptanzkriterien	Der Evaluator ist in der Lage, die normalisierten Testresultate mit den Testexpectations zu vergleichen und ein verständliches Evaluationsresultat zu erzeugen.
Aufwandschätzung Tatsächlicher Aufwand	10 Stunden 6 Stunden
Priorität	Mittel

Implementation Reporter

Beschreibung	Der Reporter nimmt die Gesamtergebnisse entgegen und gibt sie auf der Konsole in einem einfach verständlichen Format aus. Zudem schreibt er die Ergebnisse in ein Report-File, welches die Testresultate und allfällige Fehlermeldungen beinhaltet.
Akzeptanzkriterien	Die Reporter-Klasse ist in der Lage, detaillierte Berichte zu den Ergebnissen zu verfassen und abzuspeichern.
Aufwandschätzung Tatsächlicher Aufwand	10 Stunden 12 Stunden
Priorität	Niedrig

Implementation Testresultat-Mapper

Beschreibung	Der Testresultatmapper normalisiert die Rückgabewerte der einzelnen Netzwerktests in ein einheitliches Format, so dass der Evaluator damit arbeiten kann.
Akzeptanzkriterien	Die Normalform der Resultate ist festgelegt und implementiert.
Aufwandschätzung Tatsächlicher Aufwand	12 Stunden 11 Stunden
Priorität	Mittel

Implementation Testcontroller

Beschreibung	Der Testcontroller ist der Einstiegspunkt in das Programm. Er instanziert die benötigten Klassen und steuert den Programmfluss.
Akzeptanzkriterien	Testcontroller ist implementiert und das Programm lässt sich erfolgreich ausführen.
Aufwandschätzung Tatsächlicher Aufwand	4 Stunden 10 Stunden
Priorität	Niedrig

Kategorie Erstellen der Tests

Implementation Testbuilder

Beschreibung	Der Testbuilder übernimmt die Tests aus der Testdefinition und erstellt ein Testbundle mit den angegebenen Parameter.
Akzeptanzkriterien	Der Testbuilder ist implementiert und erstellt ausführbare Tests.
Aufwandschätzung Tatsächlicher Aufwand	6 Stunden 12 Stunden
Priorität	Mittel

Auswahl der Testreihenfolge

Beschreibung	Erweiterung des Testbuilder um die Möglichkeit, die Reihenfolge der Tests zu bestimmen.
Akzeptanzkriterien	Testdurchführungsreihenfolge lässt sich ändern.
Aufwandschätzung Tatsächlicher Aufwand	6 Stunden 18 Stunden
Priorität	Niedrig

Inventar Management Klassen

Beschreibung	Zu diesem Arbeitspaket gehören die Klassen Inventar, Device und DeviceConnection. Diese Klassen beschäftigen sich damit, die Informationen über die Physischen Geräte in einem Netzwerk festzuhalten.
Akzeptanzkriterien	Klassen sind implementiert und die Zusammenarbeit funktioniert. Es lassen sich Tests mit den Gerätedaten vom TestBuilder erstellen und durchführen.
Aufwandschätzung Tatsächlicher Aufwand	10 Stunden 14 Stunden
Priorität	Hoch

Testdefinition Klassen

Beschreibung	Die Testdefinitionen beschreiben die auszuführenden Tests gegen das Netzwerk. Der Loader instanziert die Definitionen der einzelnen Tests und gibt diese als Collection an den Testbuilder weiter, der daraus konkrete Tests instanziert.
Akzeptanzkriterien	Testdefinitionen lassen sich erstellen und ausführen.
Aufwandschätzung Tatsächlicher Aufwand	8 Stunden 10 Stunden
Priorität	Hoch

Connection-Strategy-Factory

Beschreibung	Dieser Teil der Software kümmert sich um die Auswahl und Instanzierung der Kommunikationsschnittstelle.
Akzeptanzkriterien	Alle Klassen sind gemäss der Architektur implementieren und es lässt sich mindestens eine Schnittstelle auswählen und instanzieren.
Aufwandschätzung Tatsächlicher Aufwand	20 Stunden 5 Stunden
Priorität	Hoch

Testerstellung-Strategy-Factory

Beschreibung	Dieser Teil des Programms entscheidet, welche konkreten Testklassen verwendet werden und instanziert diese.
Akzeptanzkriterien	Alle Klassen sind gemäss der Architektur implementiert und es gibt mindestens eine konkrete Testklasse die sich ausführen lässt.
Aufwandschätzung Tatsächlicher Aufwand	20 Stunden 24 Stunden
Priorität	Hoch

Automatische Geräteerfassung

Beschreibung	Damit das Inventar nicht von Hand geführt werden muss, lassen sich die Geräteeinstellungen automatisch erfassen.
Akzeptanzkriterien	Die Erfassung lässt sich erfolgreich durchführen, sie erfasst sämtliche Geräte im Netzwerksystem und speichert die Daten im YAML-Format.
Aufwandschätzung Tatsächlicher Aufwand	30 Stunden 0 Stunden (Wurde nicht Implementiert)
Priorität	Niedrig

Kategorie Hilfsklassen

FileHandler

Beschreibung	Der Filehandler wird immer dann aufgerufen, wenn etwas aus dem Filesystem geladen werden muss oder etwas darin geschrieben werden soll.
Akzeptanzkriterien	Der Filehandler ist implementiert und es lassen sich damit Dokumente im YAML Format schreiben und lesen.
Aufwandschätzung Tatsächlicher Aufwand	6 Stunden 10 Stunden
Priorität	Hoch

Logging

Beschreibung	Der Logger ist dafür zuständig, dass alle Events, die in der Software geschehen, beispielsweise Exceptions oder Informations-Logs, persistent gespeichert werden.
Akzeptanzkriterien	Der Logger ist implementiert.
Aufwandschätzung Tatsächlicher Aufwand	6 Stunden 5 Stunden
Priorität	Niedrig

Integration Tests

Beschreibung	Mit den Integration Tests wird überprüft, ob die Softwarekomponenten alle miteinander zusammen agieren können. Sie dienen der Überprüfung des Systems.
Akzeptanzkriterien	Integrationstests wurden systematisch und geplant durchgeführt und die Resultate dokumentiert. Allfällige Fehler wurden im Issue-Tracking erfasst.
Aufwandschätzung Tatsächlicher Aufwand	10 Stunden 5 Stunden
Priorität	Niedrig

Kommandozeilen-UI

Beschreibung	Um mit dem Programm interagieren zu können, wird ein Grafikinterface benötigt. Vorerst wird dieses mit einem Kommandozeilen-Userinterface umgesetzt.
Akzeptanzkriterien	Es ist möglich, die Erfassung, Orchestrierung und Durchführung über die Kommandozeile zu steuern.
Aufwandschätzung	14 Stunden
Tatsächlicher Aufwand	8 Stunden
Priorität	Mittel

Kategorie Test-Integration

Die Testintegration befasst sich mit der Implementation der einzelnen Netzwerktests. Für die meisten Tests sollte ein Aufwand von vier bis acht Stunden ausreichend sein, wobei umfangreichere Tests durchaus mehr Zeit für die Implementation benötigen können. Die Integration von weiteren Tests hat geringe Priorität, solange die Kernfunktionalität des Programms noch nicht fertiggestellt ist. Ausnahme dafür ist der Ping-Test, welcher für die Entwicklung zum Ausprobieren verwendet wird und ein bis drei weitere Tests, welche für das Integration-Testing verwendet werden.

Mögliche Tests:

- ping
- tracert
- show ip interface
- show ip protocols
- show ip route
- show interfaces
- iperf
- nmap portscan
- show ip ospf interface
- show ip ospf neighbor
- show ip eigrp neighbors
- show ip bgp
- show ip route bgp
- show ip bgp neighbors
- show mpls lsp extensive
- show mpls neighbor
- show isis adjacency
- show isis interface

1.5 Risikomanagement

1.5.1 Risikoanalyse

Risiken zu Beginn der Studienarbeit

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten	Risikoabdeckung
R1	Komplexität	Die als Ziel gesetzten Funktionen sind zu komplex und es müssen wesentliche Funktionalitäten weggelassen werden.	40	20%	8	Frühe Planung und Abschätzung der Machbarkeit. Regelmässige Kontrolle von Soll und Ist -Zustand	Schadensminimierung durch Abgrenzung von funktionalen und nichtfunktionalen Bestandteilen der Software.	Beschränkung auf Core-Funktionalitäten der Applikation.
R2	Geringe Erfahrung mit Python	Wir wenden zu viel Zeit an, um uns in die Programmiersprache Python einzuarbeiten	20	15%	3	Möglichst früh im Projekt in freien Zeiträumen sich einarbeiten.	In der Freizeit mit Python-Kursen oder Tutorials nochmals schulen	Die Teammitglieder haben sich bereits mit Python eingearbeitet. Es stehen umfangreiche Dokumentationen bereit, um im Fragefall schnell Lösungen zu finden
R3	Ausfall der verwendeten Hardware	Computerprobleme, Router funktionieren nicht oder andere Hardwarespezifische probleme	10	5%	0,5	Regelmässige Checks der Geräte auf Funktionalität und Gerätegesundheit	Anfordern oder organisieren von Ersatzgeräten.	Praktisch nicht möglich. Geringe Mitigation durch Arbeit mit Git und/oder in VM
R4	Qualität der Software	Die Qualität der Software ist nicht auf dem Stand, der vom Projektbetreuer gewünscht ist.	16	10%	1,6	Qualitätsmanagement während dem Projetk durchführen	Refactoring der nicht zufriedenstellenden Bestandteile	Test driven Design. Qualitätsmassnahmen werden eingehalten
R5	Probleme mit Schnittstellen	Die verwendeten Schnittstellen, z.B. Nornir/Openconnect bereiten probleme oder müssen für die Lösungserarbeitung erweitert werden	16	5%	0,8	Gute Abklärung der zu Verwendenden Tools. Prototypen erstellen, der die Grundfunktionalität aufzeigt	Frühzeitig beim Betreuer melden und eine Lösung suchen, sobald man merkt dass der Zeitplan nicht eingehalten werden kann. Funktionen weglassen.	Allfällige Erweiterungen werden früh erkannt und in der Funktionalität mit eingeplant
R6	Probleme beim Parsen	Die YAML konfiguration lässt sich nicht wie gedacht in Python einlesen und verwenden	5	10%	0,5	Im Prototypen bereits testen	Wechseln auf ein anderes Dateiformat z.B. JSON oder XML	Frühe Tests mit dem Prototypen ergeben keine Probleme
R7	Probleme mit den Rückgabewerten	Rückgabewerte der Hardwar sind ungenau oder in einem nicht verwertbaren Format	10	5%	0,5	Testen im Prototypen und studium der Dokumentationen der jeweiligen Hardware	Bei den Betreuenden Personen um Unterstützng anfragen oder Hardware Austauschen	Keine Probleme mit dem Prototypen festgestellt.
R8	Probleme mit der Erstellung der Logs/Reporting	Die Beschreibung der Logs lässt sich schwieriger Implementieren, als zu beginn angenommen	10	5%	0,5	Keine Einflussmöglichkeiten	Direkter Austausch der Commits zwischen den lokalen Repostories.	Keine Abdeckung
R9	Requirements nicht verstanden	Die Arbeit entspricht nicht den vom Kunden gewünschten Anforderungen	16	50%	8	Mittels Workshop und Requirements erfassung Risiko minimieren	Refactoring der nicht zufriedenstellenden Bestandteile	Keine Abdeckung
Summe			**143**		**23,4**			

Abbildung 2: Risikoanalyse zu Beginn der Arbeit

Angepasste Risiken am Ende der Studienarbeit

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten	Risikoabdeckung
R1	Komplexität	Die als Ziel gesetzten Funktionen sind zu komplex und es müssen wesentliche Funktionalitäten weggelassen werden.	40	5%	2	Frühe Planung und Abschätzung der Machbarkeit. Regelmässige Kontrolle von Soll und Ist -Zustand	Schadensminimierung durch Abgrenzung von funktionalen und nichtfunktionalen Bestandteilen der Software.	Beschränkung auf Core-Funktionalitäten der Applikation.
R2	Geringe Erfahrung mit Python	Wir wenden zu viel Zeit an, um uns in die Programmiersprache Python einzuarbeiten	20	10%	2	Möglichst früh im Projekt in freien Zeiträumen sich einarbeiten.	In der Freizeit mit Python-Kursen oder Tutorials nochmals schulen	Die Teammitglieder haben sich bereits mit Python eingearbeitet. Es stehen umfangreiche Dokumentationen bereit, um im Fragefall schnell Lösungen zu finden
R3	Ausfall der verwendeten Hardware	Computerprobleme, Router funktionieren nicht oder andere Hardwarespezifische probleme	10	5%	0,5	Regelmässige Checks der Geräte auf Funktionalität und Gerätegesundheit	Anfordern oder organisieren von Ersatzgeräten.	Praktisch nicht möglich. Geringe Mitigation durch Arbeit mit Git und/oder in VM
R4	Qualität der Software	Die Qualität der Software ist nicht auf dem Stand, der vom Projektbetreuer gewünscht ist.	16	15%	2,4	Qualitätsmanagement während dem Projetk durchführen	Refactoring der nicht zufriedenstellenden Bestandteile	Test driven Design. Qualitätsmassnahmen werden eingehalten
R5	Probleme mit Schnittstellen	Die verwendeten Schnittstellen, z.B. Nornir/Openconnect bereiten probleme oder müssen für die Lösungserarbeitung erweitert werden	16	5%	0,8	Gute Abklärung der zu Verwendenden Tools. Prototypen erstellen, der die Grundfunktionalität aufzeigt	Frühzeitig beim Betreuer melden und eine Lösung suchen, sobald man merkt dass der Zeitplan nicht eingehalten werden kann. Funktionen weglassen.	Allfällige Erweiterungen werden früh erkannt und in der Funktionalität mit eingeplant
R6	Probleme beim Parsen	Die YAML konfiguration lässt sich nicht wie gedacht in Python einlesen und verwenden	5	10%	0,5	Im Prototypen bereits testen	Wechseln auf ein anderes Dateiformat z.B. JSON oder XML	Frühe Tests mit dem Prototypen ergeben keine Probleme
R7	Probleme mit den Rückgabewerten	Rückgabewerte der Hardwar sind ungenau oder in einem nicht verwertbaren Format	10	5%	0,5	Testen im Prototypen und studium der Dokumentationen der jeweiligen Hardware	Bei den Betreuenden Personen um Unterstützng anfragen oder Hardware Austauschen	Keine Probleme mit dem Prototypen festgestellt.
R8	Probleme mit der Erstellung der Logs/Reporting	Die Beschreibung der Logs lässt sich schwieriger Implementieren, als zu beginn angenommen	10	5%	0,5	Keine Einflussmöglichkeiten	Direkter Austausch der Commits zwischen den lokalen Repostories.	Keine Abdeckung
R9	Requirements nicht verstanden	Die Arbeit entspricht nicht den vom Kunden gewünschten Anforderungen	16	10%	1,6	Mittels Workshop und Requirements erfassung Risiko minimieren	Refactoring der nicht zufriedenstellenden Bestandteile	Keine Abdeckung
Summe			**143**		**10,8**			

Abbildung 3: Angepasste Risiken am Ende der Studienarbeit

1.5.2 Umgang mit Risiken

Um Probleme gerade während der Init/Analyse Phase möglichst früh zu erkennen, arbeiten wir wöchentlich zwei Tage nebeneinander, um uns über mögliche Probleme auszutauschen. Des Weiteren suchen wir auch den Kontakt zum Betreuer sobald Unklarheiten im Team herrschen. Einmal alle zwei Wochen sind am Samstag für drei bis vier Stunden vorgesehen, um an der Dokumentation und and der Reduktion aufgetretener Risiken zu arbeiten.

Darüber hinaus ist in der Zeitplanung keine Reserve für Risiken eingeplant und die Projektmitglieder werden im Eintretensfall zusätzliche Arbeit verrichten müssen, um die verlorene Zeit aufzuholen.

2 Anforderungen

Im folgenden Abschnitt werden die Anforderungen an ein Netzwerk-Test-System formuliert. Es werden die Kern-Akteuere identifiziert und deren Funktion und Abhängigkeiten und Anforderungen formuliert, um auf dieser Basis die Software zu entwerfen.

2.1 Akteure in einem Netzwerksystem

In der Praxis gibt es für die verschiedenen Akteure in einem Netzwerksystem unterschiedliche Bezeichnungen. Beispielsweise ist oft nicht klar, was der Unterschied zwischen einem Netzwerk-Architekten und einem Netzwerk-Engineer ist und welche Verantwortungen diese nun genau haben. Wir haben eine eigene Unterscheidung der Akteure formuliert und diese in den kommenden Sektionen dokumentiert, um eine einheitliche Basis für die Leser zu schaffen.

Netzwerk-Architekt

Ein Netzwerk-Architekt plant und erstellt Kommunikationsnetzwerke. Im Zuge dieser Arbeit wurde zwischen dem Architekten als verantwortlichen Senior-Network-Engineer und einem Network Engineer (Junior oder Senior) als operativen Mitarbeiter unterschieden. Der Architekt nimmt dabei eher die Rolle des Managers oder Teamleiters ein. Er führt dabei normalerweise keine Konfigurationen am Netzwerk durch.

Netzwerk-Engineer

Ein Netzwerk-Engineer ist für die Installation und Instandhaltung eines Netzwerks zuständig. Er ist dem Netzwerk-Architekten unterstellt und setzt mit Ihm zusammen die geplanten Arbeiten um.

Netzwerk-Administrator

er Netzwerk Administrator hat üblicherweise eine abgeschlossene Berufslehre in der Informatik und arbeitet zusammen mit dem Netzwerk-Engineer am Netzwerk. Es wird davon ausgegangen, dass ein Netzwerk Administrator keine bis wenige Programmierkenntnisse hat. Ein Netzwerk Administrator hat, je nach Grösse des Netzwerks, nur Kenntnisse über einen Teil der Netzwerkumgebung. Er führt dabei ihm vom Architekten oder Engineer vorgegebene Arbeiten aus und muss dazu nicht den vollen Überblick über das Netzwerk und die darin verwendeten Technologien haben.

Netzwerk-User

Benutzer der Netzwerkumgebung. User können das Netzwerk verwenden, aber nicht dessen Konfigurationen anpassen.

Netzwerk-Gerät

Ein Netzwerkgerät kann aus Hardware wie Switch, Router oder Server bestehen oder Virtuell als Software implementiert sein. Im Zuge der Arbeit werden Netzwerkgeräte auch als Netzwerk-Devices oder einfach Device bezeichnet. Typischerweise haben Devices eine Statische Konfiguration und einen dynamischen Zustand zur Laufzeit. In den kommenden Kapiteln wird genauer auf Netzwerkgeräte eingegangen.

Netzwerk-Verbindung

Die Netzwerkverbindung ist der Kommunikationskanal zwischen den einzelnen Netzwerkgeräten. Sie kann in physischer Form als Kabel, oder mit kabellosen Mitteln z.B. Funk umgesetzt sein. Die Wahl des Übertragungsmediums hat grossen Einfluss über die verfügbare Bandbreite und mögliche Störfaktoren.

Repository/Inventar

Im Inventar werden die Unterschiedlichen Devices mit den für den Betrieb wichtigsten Parametern gespeichert. Das Inventar kann in digitaler Form als Repository, als File auf einem Ordner/Computer, oder analogin einem Dokumentenorder abgelegt sein. Das Inventar wird benötigt, um die aktuellen Konfigurationen, die physische Position des Geräts oder sonstige für den Betrieb relevanten Informationen zu dokumentieren.

2.2 Akteure in der zu entwickelnden Software

Testprogramm

Das Testprogramm ist der Kern des zu entwickelnden Systems dieser Arbeit. Es Interagiert mit den anderen Akteuren und hat, vom Akteur und Kontext abhängig, unterschiedliche Anforderungen. Es soll so aufgebaut sein, dass ein Benutzer der Software diese mit möglichst geringem Aufwand bedienen kann.

Testdefinitionssprache

Wird im Rahmen dieser Arbeit auch als Testbeschreibungssprache oder Definitionssprache bezeichnet. Eine Testdefinition beschreibt die einzelnen Testfälle, die von einem System durchgeführt werden sollen. Die Definitionssprache soll dabei in einem Format gehalten werden, das von allen Benutzern der Software verstanden wird und von diesen erweitert werden kann.

Testreport

Ein Testreport soll, möglichst einfach und genau, die Ergebnisse eines Netzwerktests aufzeigen. Fehlgeschlagene Tests sollen dabei möglichst einfach und schnell zu erkennen sein und alle Informationen beinhalten, die ein Betrachter benötigt, um den Fehler im System zu lokalisieren und beheben. Ausserdem muss mindestens noch ein Zeitstempel vorhanden sein, um die Historie vergangener Netzwerktests nachvolziehen zu können. Testreporte können auf dem System, welches die Tests ausführt, oder in einem zentralen Repository abgelegt werden, damit mehrere Mitglieder eines Netzwerkteams gleichzeitig darauf zugreifen können.

Kommunikationskanal

Der Kommunikationskanal, nicht zu verwechseln mit der Netzwerkverbindung zwischen zwei Netzwerkgeräten, verbindet ein zu testendes Netzwerk mit dem Testprogramm. Möglichkeiten für einen Kanal sind beispielsweise das SSH (secure shell) Protokoll oder der Restconf Standard. Dies kann über eine Kabelverbindung oder Kabellos geschehen. Die Wahl des Kommunikationskanals beeinflusst dabei, in welcher Form Netzwerktests durchgeführt werden können und in welchem Format die Ergebnisse zurückgegeben werden.

Netzwerktest

Werden heute meist manuell oder mit Hilfe eines Skripts durchgeführt. Ein automatisierter Netzwerktest sollte hypothetisch ad-hoc nach jeder Konfigurationsänderung vom Testprogramm durchgeführt werden um die Funktionsweise des Netzwerks zu validieren. Ein Benutzer des zu entwickelnden Systems soll in der Lage sein, mit nur geringer Einarbeitungszeit, Netzwerktests zu spezifizieren und durchzuführen.

2.3 Use Cases

2.3.1 Use Case Diagramm

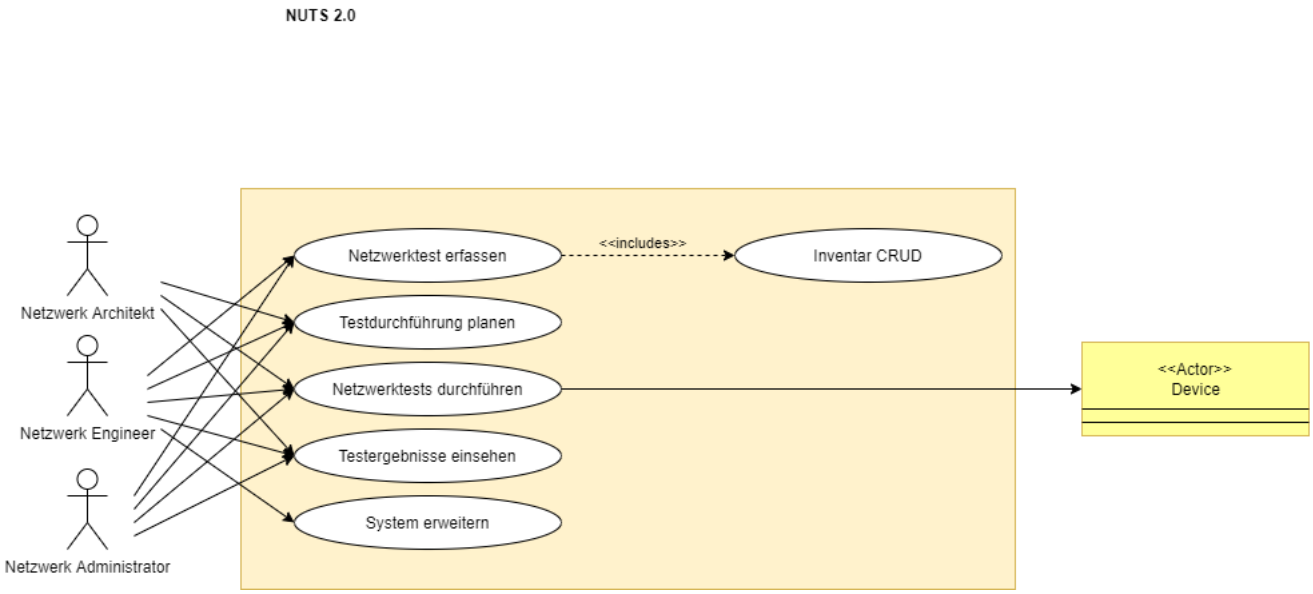

Abbildung 4: Use Case Diagramm

2.3.2 Aktoren

Die Primären Akteure sind der Netzwerk Architekt, -Administrator und -Engineer. Der Architekt will primär die Ergebnisse einsehen können, um zu sehen, dass das Netzwerk korrekt funktioniert. Ausserdem möchte er, beispielsweise um eine Erweiterung des Netzwerks zu Planen, eine Ausführung von Netzwerktests konfigurieren und durchführen. Der Administrator will Netzwerktests erfassen, deren Durchführung planen, die Tests durchführen und die Ergebnisse einsehen. Der Engineer möchte neben den Tätigkeiten, die der Administrator ausführt, zudem das Testsystem um weitere Netzwerktests erweitern können.

2.4 Beschreibung Usecases (Brief)

Netzwerktest erfassen Ein Netzwerktest setzt sich zusammen aus der Testdefinition mit den zu testenden Devices, Befehle, die auf den Devicesv ausgeführt werden sollen, einem oder mehreren Kommunikationskanälen, über den die Devices angesprochen werden und einem Erwartungswert für das Ergebnis. Diese Informationen werden in einer Testdefinitionssprache gespeichert, welche so strukturiert sein muss, dass sie ein Softwaresystem einfach laden kann und trotzdem von Menschen interpretiert werden kann. Die Erfassung eines Netzwerktests soll so einfach wie möglich gehalten werden, damit Administratoren und Engineers effizient neue Tests spezifizieren können.

Inventar CRUD Der Netzwerk Engineer oder -Administrator möchte die physischen und virtuellen Netzwerkgeräte und deren statische Konfiguration in einem Inventar verwalten. Das Inventar wird von dem zu entwickelnden System verwendet, um die Gerätekonfigurationen wie z.B. Zugangsdaten oder Herstellerinformationen abzurufen. Es soll möglich sein, das Inventar automatisiert zu erstellen und Geräte in distinkte Gruppen zu kategorisieren.

Testdurchführung planen Themen, die in der Testausführung relevant sind, sind die Auswahl, welche Tests überhaupt ausgeführt werden sollen, die Reihenfolge der Tests, auf welchen Teil des Systems sie angewandt werden sollen und ob sie synchron oder asynchron durchgeführt werden. Weitere Punkte wären das automatische durchführen von Tests zu spezifischen Zeiten oder Wochentagen und dass die Testkonfiguration gespeichert wird, um sie später anzupassen. Auch hier ist darauf zu achten, dass das zu entwickelnde System so aufgebaut ist, dass Engineers, Architekten und Administratoren effizient arbeiten können.

Netzwerktests durchführen Der Anwender möchte die in der Testdurchführung geplanten Netwerktests auf das angegebene System ausführen. Dazu wird in einer Benutzeroberfläche die Testausführung gestartet oder auf einem Gerät automatisch die zu entwickelnde Software ausgeführt.

Testergebnisse einsehen Nachdem ein Test durchgeführt wurde, soll ein Testresultat angezeigt werden. In den Resultaten soll ersichtlich sein, welche Tests durchgeführt wurde, was der Test genau gemacht hat, welche Befehle auf welchen Devices ausgeführt wurde und wie das Ergebnis ist. Die Testergebnisse sollen auf der Konsole/Benutzeroberfläche ersichtlich sein und zusätzlich in einem Testreport mit Datum und Uhrzeit gespeichert werden, damit die Historie des Netzwerks ermittelt werden kann. Wenn Tests durch einen Fehler im zu entwickelnden System nicht durchgeführt werden kann, sollen alle anderen Tests nicht davon beeinflusst werden und das Ergebnis soll einen Vermerk für das Versagen des Systems beinhalten, mit dem der Anwender in der lage ist, die Ursache zu ermitteln und zu beheben.

System erweitern Engineers sollen in der Lage sein, das System bei Bedarf zu erweitern, z.B. um weitere Tests oder Netzwerkschnittstellen hinzuzufügen oder Fehler zu verbessern. Die Erweiterungen beschränken sich aber auf rein funktionale Bereiche des Systems. Für Änderungen an der Benutzeroberfläche sollen Softwareengineers mit Erfahrung auf dem Gebiet hinzugezogen werden.

2.5 Nichtfunktionale Anforderungen

In diesem Kapitel werden die nichtfunktionalen Anforderungen an das Projekt behandelt. Es werden Aspekte und Anforderungen aus den Bereichen Änderbarkeit, Benutzbarkeit, Effizienz, Zuverlässigkeit, Betreibbarkeit und Sicherheit gemäss ISO/IEC 9126 betrachtet. Die jeweiligen Aspekte werden in ihren Unterkapiteln genauer beschrieben. Es wurden mögliche Szenarien erarbeitet, die in der Erstellung oder dem Betrieb der Software auftreten können und beim Architekturdesign in betracht gezogen wurden.

2.5.1 Änderbarkeit

Aufwand, der zur Durchführung von vorgegebenen Änderungsarbeiten benötigt wird. Unter Änderungen gehen Korrekturen, Anpassungen oder Veränderungen der Umgebung, Anforderungen oder funktionalen Spezifikation. Gemäss ISO 9126 gehören zur Änderbarkeit folgende Teilmerkmale:

Analysierbarkeit Aufwand, der benötigt wird, um das System zu verstehen, z.B. um Ursachen von Versagen oder Mängel zu diagnostizieren oder Änderungen zu planen.

Modifizierbarkeit Wie leicht lässt sich das System anpassen, um Verbesserungen oder Fehlerbeseitigungen durchzuführen.

Stabilität Wahrscheinlichkeit, dass mit Änderungen unerwartete Nebenwirkungen auftreten.

Testbarkeit Wie gross wird der Aufwand, bei Änderungen die Software zu prüfen.

Szenario: Neue Netzwerkschnittstelle Wenn zum bestehenden System eine neue Netzwerkschnittstelle definiert werden soll, so muss die dafür notwendige Software innerhalb von einer Arbeitswoche entwickelt, integriert und in Betrieb genommen werden können.

Qualitätsziele	Flexibilität, Erweiterbarkeit, Anpassbarkeit, Austauschbarkeit
Geschäftsziel(e)	Software kann mit geringem Aufwand an geänderte Anforderungen angepasst werden
Auslöser	Ein Engineer möchte weitere Tests einbinden oder Schnittstellen, die nicht im System integriert sind.
Reaktion	Die Software lässt sich von einem Entwickler in weniger als einer Woche um benötigte Komponenten erweitern.
Zielwert	Erweiterungen der Netzwerkschnittstellen oder Anpassungen von Tests sind innerhalb von 40 Personenstunden umsetzbar.

Tabelle 6: Szenario: Neue Schnittstelle

Scenario: Schnelle Fehlerlokalisierung Die Ursache von fehlgeschlagenen Tests (Software-Unittests) lässt sich in kurzer Zeit lokalisieren.

Qualitätsziele	Schnelle Fehlerbehebung, Änderbarkeit, Anpassbarkeit, geringes Risiko bei Erweiterungen
Geschäftsziel(e)	Entwickler können das Programm einfach anpassen und erkennen im Fehlerfall schnell, was nicht funktioniert hat.
Auslöser	Eine Änderung im Code führt zu Fehlnern in der Ausführung.
Reaktion	Wenn ein Fehler dazu führt, dass die Softwareausführung fehlschlägt, kann ein Entwickler aufgrund von Fehler- und/oder Log-Nachrichten die Ursache in kurzer Zeit lokalisieren.
Zielwert	Fehlerlokalisierung findet durchschnittlich in weniger als 10 Minuten statt.

Tabelle 7: Szenario: Schnelle Fehlerlokalisierung

2.5.2 Benutzbarkeit

Zeitlicher Aufwand, der für die Erlernung der Benutzung des Programms benötigt wird. Die User werden hierfür in spezifische Nutzergruppen mit festgelegten Fähigkeiten unterteilt.

Verständlichkeit Aufwand für den Nutzer, die Konzepte und Menüführung der Anwendung zu verstehen.

Erlernbarkeit Aufwand für den User, sich ohne Vorwissen in das System einzuarbeiten.

Bedienbarkeit Aufwand für den Benutzer, die Anwendung zu bedienen.

Szenario: Einfachheit der Definitionssprache Die Definitionen von Inventar und Tests sind so aufgebaut, dass ein User in kurzer Zeit die Struktur und den Aufbau versteht und eigene Tests implementieren kann.

Qualitätsziele	Produktivität, Einfachheit, Verständlichkeit
Geschäftsziel(e)	Einarbeitung in die Testdefinition erfolg möglichst einfach und benötigt nur geringes Vorwissen.
Auslöser	Ein Nutzer, welcher keine Erfahrung im Umgang mit der Software hat, möchte eigene Tests definieren.
Reaktion	Benutzer können sich schnell in die Testdefinitionen einlesen und rasch eigene Tests definieren, vorausgesetzt, sie haben Kenntnisse des Netzwerkes.
Zielwert	Ungeschulte Nutzer verstehen innerhalb von durchschnittlich 30 Minuten die Struktur und den Aufbau der Testdefinitionen und sind in der Lage, eigene Tests zu erstellen.

Tabelle 8: Szenario: Einfachheit der Definitionssprache

Szenario: Hinweis auf Fehleingaben Fehlerhafte Eingaben werden vom System ignoriert und der Benutzer wird auf die falsche Eingabe hingewiesen. Das Programm führt fehlerfreie Programmteile unabhängig von den Fehlern durch.

Qualitätsziele	Robustheit, Verständlichkeit, Fehlertoleranz.
Geschäftsziel(e)	Fehleingaben führen nicht dazu, dass die Tests nicht mehr durchgeführt werden können.
Auslöser	Ein Benutzer macht einen Fehler bei der Testdefinition und startet das Programm.
Reaktion	Das Programm führt alle korrekten Tests durch und informiert den Benutzer, dass es fehlerhafte Tests gibt, die nicht durchgeführt werden können. Die Hinweise werden im Report und auf der Konsolenausgabe geschrieben.
Zielwert	Tests sind einzeln gekapselt und werden unabhängig voneinander durchgeführt. Falscheingaben werden vom Programm detektiert und im Testreport sowie auf der Konsolenausgabe erwähnt.

Tabelle 9: Szenario: Hinweis auf Fehleingaben

2.5.3 Effizienz

Mit Effizienz ist die 'performance efficiency' gemeint, d.h. das Verhältnis zwischen dem Leistungsniveau der Software und den eingesetzten Hardwarekomponenten. Andere Beschreibungen umfassen: Skalierbarkeit, Speicherbedarf, Verarbeitungsgeschwindigkeit, Antwortzeit etc. Teilmerkmale nach ISO 9126:

Zeitverhalten Dauer für Verarbeitung und Antwortzeit sowie Durchsatz bei der Ausführung des Programms

Verbrauchsverhalten Wie viel Speicherbedarf hat das Programm, wie lange werden Betriebsmittel in Anspruch genommen und welche Hardwarekomponenten werden benötigt.

2.5.4 Zuverlässigkeit

Unter Zuverlässigkeit versteht man die Fähigkeit der Software, unter festgelegten Bedingungen die Funktionalität über einen definierten Zeitraum zu gewährleisten

Reife Geringe Ausfallhäufigkeit durch Fehlzustände.

Fehlertoleranz Die Software ist in der Lage, trotz Fehlern ihr spezifiziertes Leistungsniveau beizubehalten.

Wiederherstellbarkeit Im Fehlerfall können betroffene Daten wiederhergestellt und die Funktionalität wieder aufgenommen werden.

Szenario: Tests lassen sich auf der Netzwerkseite nicht ausführen Falls ein Test auf dem jeweiligen Netzwerkgerät nicht erfolgreich durchgeführt werden kann, läuft das Programm weiter und definiert den dazugehörigen Netzwerktest als nicht bestanden.

Qualitätsziele	Robustheit, Behandlung Infrastrukturbedingter Fehler.
Geschäftsziel(e)	Das System führt alle Tests unabhängig voneinander durch. Wenn ein Test zu einem Fehler führt, weil z.B. ein falsches Netzwerkgerät angegeben wurde, wird dieser Test unabhängig von allen anderen Tests fehlschlagen.
Auslöser	Test lässt sich auf spezifizierter Infrastruktur nicht ausführen.
Reaktion	Test schlägt fehl und mögliche Ursachen werden im Report und in der Konsole angezeigt. Alle anderen Tests laufen durch.
Zielwert	Das Fehlschlagen eines Tests fürht nicht zum Programmabbruch.

Tabelle 10: Szenario: Testausführung Netzwerkseitig nicht ausführbar

2.5.5 Betreibbarkeit

Die Betriebbarkeit wird in der ISO 9126 nicht definiert. Die ISO spezifiziert aber mehrere Teilmerkmale, die unter dem Begriff Betreibbarkeit zusammengefasst werden können:

Analysierbarkeit Aufwand, der benötigt wird, um den Code zu analysieren, um im falle eines Versagens dessen Ursachen zu diagnostizieren oder um Änderungen zu planen und durchzuführen.

Installierbarkeit Aufwand, das Programm auf einem frisch aufgesetzten Gerät laufen zu lassen.

Übertragbarkeit Kann die Software von einer Umgebung auf eine andere übertragen werden. Als Umgebung zählen Hardwarekomponenten, Softwarekomponenten, Organisatorische Umgebungen oder Betriebssysteme.

Austauschbarkeit Aufwand und Möglichkeit, die Software anstelle einer anderen in deren spezifizierten Umgebung laufen zu lassen.

Koexistenz Fähigkeit der Software, neben anderen Programmen mit ähnlichen oder übereinstimmenden Funktionen zu arbeiten.

Szenario: Einfache Installation auf einem neuen Gerät Das Programm lässt sich auf einem neuen Gerät ohne grossen Mehraufwand installieren, ohne dass die Funktionalität des Geräts beeinflusst wird.

Qualitätsziele	Einfachheit, Portierbarkeit, Benutzbarkeit
Geschäftsziel(e)	Die Installation der Software ist so einfach, dass sie innert kurzer Zeit und/oder automatisiert durchgeführt werden kann.
Auslöser	Die Testsoftware soll auf einem frisch aufgesetzten Gerät installiert werden.
Reaktion	Installationszeiten sind gering, benötigen wenige bis keine weiteren Softwarekomponenten oder lässt sich mit einigen Kommandozeilenbefehlen automatisch installieren.
Zielwert	Die Software wird mit einer Installationsanleitung ausgeliefert, die einfach und verständlich die Inbetriebnahme des Programms erklärt. Abhängigkeiten zu anderen Softwarekomponenten werden bewusst gering gehalten um eine einfache Installation mit weniger als 30 Minuten Zeitaufwand zu gewährleisten.

Tabelle 11: Szenario: Einfache Installation auf einem anderen Gerät

2.5.6 Sicherheit

In dieser Sektion werden Sicherheitsanforderungen beschrieben. Verschlüsselung, Privacy und der Umgang mit Passwörtern.

Verschlüsselung von Datenübertragungen Die Netzwerktest werden über eine Verschlüsselte Verbindung durchgeführt, die dem aktuellen Stand der Technik entspricht.

Umgang mit Passwörtern Zugangsdaten der Devices werden im Inventar in Unverschlüsselter Form abgelegt. Es liegt in der Verantortung der Betreiber des Netzwerks, dass die Zugangsdaten nicht von dritten eingesehen werden.

3 Design

3.1 Domänenmodell

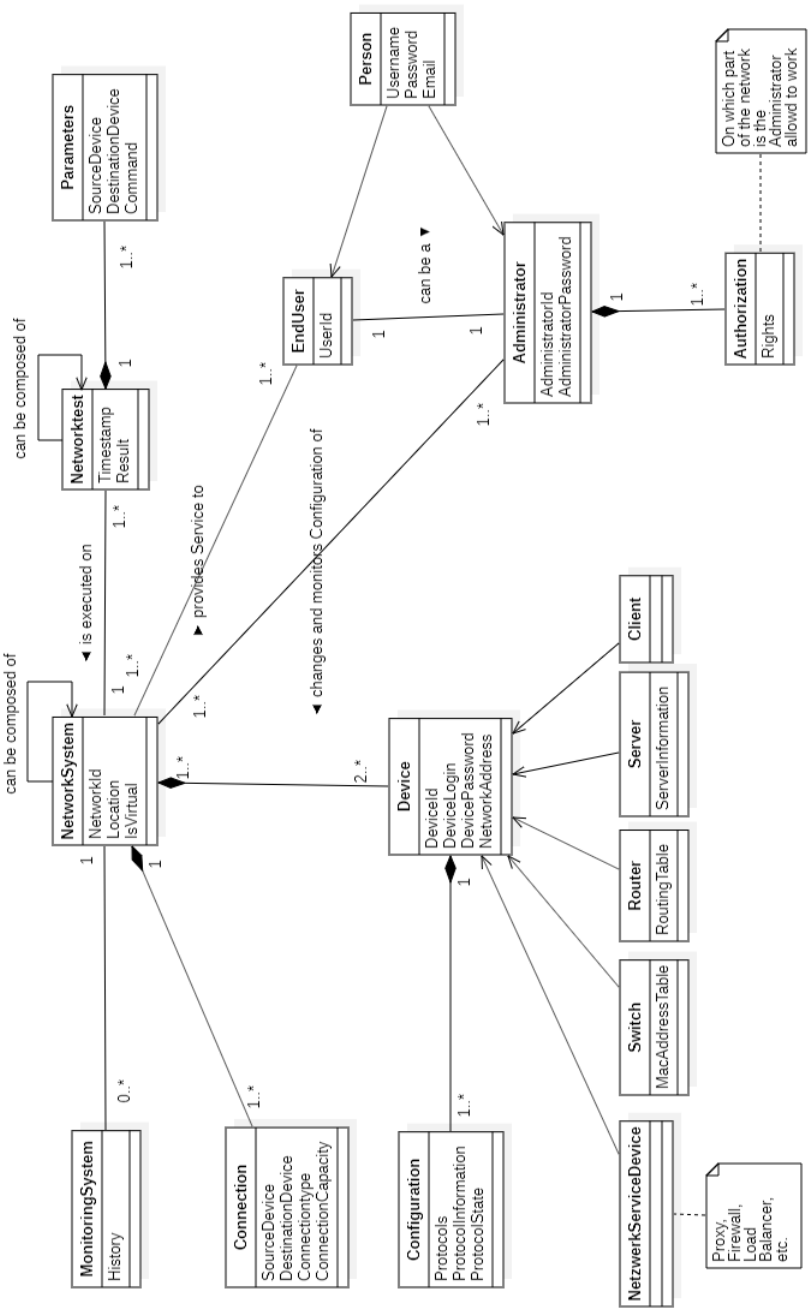

Abbildung 5: Domainmodell

3.1.1 Prosa

Ein Netzwerk (Network System) setzt sich aus mindestens zwei Geräten (Device) und Verbindungen dazwischen (Connection) zusammen. Es kann auch mehrere Teilnetzwerke in sich vereinen, z.B die beiden Netzwerke aus Haupt- und Nebengebäude ergeben das gesamte Firmennetzwerk. Das Netzwerksystem kann auch in virtueller Form aufgebaut sein, Beispielsweise als Netz von virtuellen Routern auf einem Server. Ein Device kann in die vier Kategorien Switch, Router (Level-3-Switch), Server und Client eingeteilt werden und hat eine oder mehrere Konfigurationen. Beispielsweise kann ein Router das OSPF (Open Shortest Path First) Protokoll und zusätzlich als Fallback statische Routen konfiguriert haben.

Geräte haben eine Identifikation, ein Gerätelogin und -passwort und eine Addresse innerhalb des Netzwerkes. Die Konfiguration der Geräte setzt sich zusammen aus dem Protokoll, dessen Informationen und dem Zustand, in dem das Gerät mit dem Protokoll gerade ist.

Es ist möglich, dass ein Netzwerksystem ein Monitoring System hat, das die Zustände zur Laufzeit überwacht, z.B. Welche Verbindungen gerade aktiv sind und wieviele Client auf dem Netzwerksystem angemeldet sind. Das Monitoring speichert auch vergangene Daten in einer Historie, so dass vergangene Zustände mit dem aktuellen Zustand verglichen werden können.

Netzwerktests, wie sie von Netzwerk-Technikern auf dem Netzwerk ausgeführt werden, haben Parameter und können aus weiteren Tests zusammengesetzt sein. Ein umfangreicher Systemtest kann aus mehreren Ping-Tests bestehen. Ein Netzwerktest umfasst einen Zeitstempel, um die genaue Durchführungszeit ermitteln zu können und hat ein Resultat, üblicherweise bestanden oder durchgefallen. Die Parameter eines Netzwerktests sind üblicherweise ein Befehl (Ping, Traceroute etc.) und ein Source- und/oder Destination-Device.

Auf dem Netzwerksystem arbeiten verschiedene Personen. Jede Person benötigt einen Usernamen, ein Passwort und eine E-Mail-Addresse, um sich gegenüber dem Neztwerk zu authentifizieren. Diese können in User und Administratoren eingeteilt werden. Usern werden Services vom Netzwerk wie Internet oder Serverzugriff angeboten, und sie haben eine UserID, mit der sie das Netzwerk erkennt. Administratoren können die Konfiguration des Netzwerks einsehen und verändern. Zusätzlich zum regulären Login haben sie einen Administratorzugriff, der aus einer ID und einem Admin-Passwort besteht. Dazu benötigen sie eine Authorisierung, um auf dem Netzwerksystem zu arbeiten.

3.2 Klassendiagramm

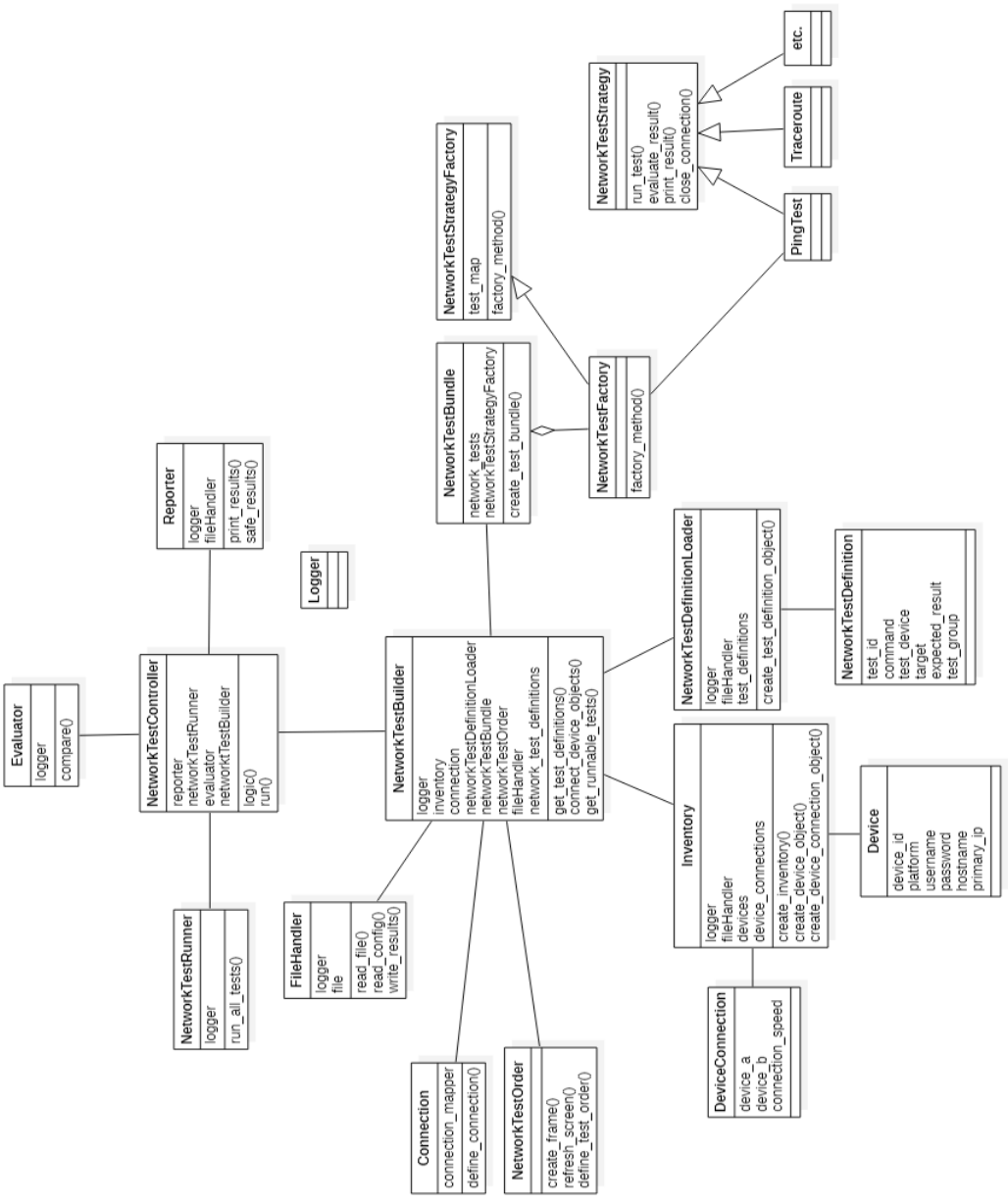

Abbildung 6: Klassendiagramm

3.2.1 Beschreibungen

NetworkTestController

Der TestController ist das Kernstück des Programms. Er beinhaltet die Main-Methode und steuert den Ablauf des Programms. Vom TestController werden der TestRunner, der TestBuilder, der Evaluator und der Reporter instanziert und er ist zuständig für die Kommunikation zwischen diesen Komponenten.

Komponenten	Beschreibung
reporter	Referenz auf die Reporter-Instanz
testRunner	Referenz auf die TestRunner-Instanz
evaluator	Referenz auf die Evaluator-Instanz
testBuilder	Referenz auf die TestBuilder-Instanz
logic()	Methode, die für die Programmausführung sämtliche referenzierten Komponenten instanziert und deren Funktionalität in der korrekten Reihenfolge ausführt.
run()	Main-Methode, die den Testcontroller instanziert.

NetworkTestRunner

Der TestRunner führt die ihm vom Controller mitgeteilten Tests gemäss der, in der TestSuite angegebenen, Parameter aus. Die zu verwendende Netzwerkschnittstelle (Restconf, Netconf, SSH etc.) wird ihm ebenfalls vom Controller mitgeteilt. Die Resultate der Netzwerktests gibt er dem TestController in Form von Rückgabewerten zurück.

Komponenten	Beschreibung
logger	Referenz auf die Logger-Instanz
runAllTests()	Methode, die die in der Testsuite spezifizierten Tests auf der in der Connection spezifizierten Netzwerkschnittstelle ausführt.

TestResultMapper

Da die Ergebnisse der Netzwerktests, je nach Betriebssystem des Neztwerkgeräts und Typ der Verbindung (Netconf, Restconf, NAPALM), ein anderes Rückgabeformat hat, müssen die Testresultate normalisiert werden. Der TestResultMapper bringt die verschiedenen Rückgabeformate in ein einheitliches Format, welches dem Evaluator für den Ist-, Soll-Vergleich übgeben wird.

Komponenten	Beschreibung
connectionType	Der Typ der Connection, mit welchem auf das Netzwerk verbunden wird.
logger	Referenz auf das Logger-Objekt.
testResult	Die Resultate, welche der TestRunner zurück gibt.
mapTestResult()	Methode, die die Resultate des TestRunners in eine einheitliche Norm bringt.

Reporter

Der Reporter schreibt die Testergebnisse der fertig ausgeführten Tests auf die Konsole/Benutzeroberfläche und erstellt ein Testprotokoll, welches er über den FileHandler im Directory abspeichert.

Komponenten	Beschreibung
evaluationResult	Die Ergebnisse des Evaluators, wie der Soll- Ist-Vergleich der Tests abgelaufen ist.
fileHandler	Referenz auf das FileHandler-Objekt.
printResults()	Methode die die Testergebnisse auf der Konsole ausgibt.
safeResults()	Methode, die über den FileHandler die Testergebnisse in einem File im Directory abspeichert.

Evaluator

Der Evaluator vergleicht die ihm vom Controller mitgeteilten Testresultate mit den Testerwartungswerten und evaluiert, ob die Tests bestanden sind oder nicht.

Komponenten	Beschreibung
testRunnerResult	Ergebnisse der auf dem Netzwerk vom Runner ausgeführten Tests.
testExpectation	Zu erwartende Ergebnisse für einen spezifischen Test der Testsuite.
logger	Referenz auf das Logger-Objekt.
evaluationResult	Resultat des Soll-Ist-Vergleichs, welches dem Controller zurückgegeben wird.
compare()	Methode, die testRunnerResult mit testExpectation vergleicht und entscheidet, ob der Test erfolgreich war, oder gescheitert ist.

FileHandler

Der FileHandler ist eine Utility-Klasse, die für das Einlesen und Schreiben von Daten aus dem Programm in das Directory zuständig ist.

Komponenten	Beschreibung
logger	Referenz auf das Logger-Objekt.
file	Pfad zu dem zu lesenden/schreibenden File.
fileType	Typ des Files, YAML, XML, JSON usw.
readFile()	Methode, die ein vorgegebenes File öffnet und deren Inhalt in das Programm einliest.
writeFile()	Methode, die einen vorgegebenen Text aus dem Programm in ein File im Directory schreibt.

Testbuilder

Der Testbuilder ist für die Zusammenstellung der Tests verantwortlich. Er instanziert einen TestDefinitionLoader, der aus dem TestDefinitionsDirectory, worin mehrere Files mit TestDefinitionen gespeichert sind, die einzelnen TestDefinitionen ausliest. Dann holt er sich aus dem Inventory die Devices und deren Connections. Die TestDefinitionen attributisiert er mit den, in der TestStrategy definierten Tests, und erstellt mit den Parametern aus dem Inventar das TestBundle. Hier kann vom Benutzer auch die Auswahl der einzelnen Tests, welche durchgeführt werden sollen, sowie die Durchführungsreihenfolge festgelegt werden. Die Connection spezifiziert die konkrete Netzwerkschnittstelle, über welche die Netzwerktests vom Runner dann durchgeführt werden sollen.

Komponenten	Beschreibung
testBundle	Collection von Tests mit Parametern für die Ausführung als Referent auf ein TestBundle-Objekt.
connection	Referenz auf das Connection-Objekt.
inventory	Referenz auf das Inventory-Objekt.
testDefinitionLoader	Referenz auf ein TestDefinitionLoader-Objekt.
testOrder	Definition, in welcher Reihenfolge die Tests ausgeführt werden sollen.
logger	Referenz auf das Logger-Objekt.
testDefinitions	Collection mit Referenzen auf TestDefinition-Objekte.
createTestSuite()	Methode, die aus den destDefinitionen, dem Inventar und der Testreihenfolge eine TestSuite erstellt.
defineTestOrder()	Methode, die die Testreihenfolge vom Softwareuser einstellen lassen kann.

TestBundle

Das TestBundle ist dafür zuständig, gemäss der TestDefinitionen die Tests zusammenzustellen. Dazu wird ein TestContext und eine TestFactory instanziert und damit die Tests ausgewählt und instanziert.

Komponenten	Beschreibung
tests	Collection von Tests, die von der TestFactory instanziert wurden und dem TestBuilder als Rückgabewert zurückgegeben wird.
testContext	Referenz auf das TestContext-Objekt.
logger	Referenz auf das Logger-Objekt.
testFactory	Referenz auf das TestFactory objekt.
createTestBundle()	Methode, die aus den TestDefinitionen über eine TestFactory die konkreten Tests auswählt und in einer Collection zusammenfasst.

TestContext

Der TestContext wird benötigt, um unter Anwendung des Strategy-Pattern die Auswahl der Tests durchzuführen. Die Methode setText() ruft dabei die Factory auf und instanziert die konkreten Tests.

TestStrategy

Das TestStrategy-Interface dient als Basis für die konkreten Implementationen der Tests. Das Interface gibt den Test-Classes dafür die benötigte Funktionalität vor, die wiederum von den konkreten Tests implementiert werden müssen. Die ExecuteTest() Methode ist ein Beispiel für eine solche Funktionalität.

TestFactory

Die TestFactory ist die Anwendung des Factory Method Pattern, welches den Tests erlaubt, instanziert zu werden, ohne dass sich diese selbst um die Instanzierungslogik kümmern müssen. Die Factory entscheidet dabei, welche Tests für welche Definitionen instanziert werden müssen und instanziert diese konkreten Tests mit der getTest() Methode.

TestDefinitionLoader

Der TestDefinitionLoader ist dafür zuständeg, die TestDefinitionen aus dem Directory zu laden und als einzelne TestDefinitionen zu instanzieren.

Komponenten	Beschreibung
testDefinitions	Collection von TestDefinitionen, die dem TestBuilder als Rückgabewerte zurückgegeben werden.
fileHandler	Referenz auf das FileHandler-Objekt.
createTestDefinitionObject	Methode, die die TestDefinitionen aus dem FileSystem ausliest und TestDefinition-Objekte für jede Definition instanziert.

Inventory

Das Inventory ist diejenige Klasse, die im Programm die einzelnen Geräte und deren Verbindungen untereinander verwaltet. Sie liest dazu aus dem FileSystem die spezifizierten Devices und Device-Connections ein und instanziert Klassen, um diese als Collection dem TestBuilder zurückzugeben.

Komponenten	Beschreibung
devices	Collection der eingelesenen Geräte.
deviceConnection	Collection der Geräteverbindungen.
fileHandler	Referenz auf das FileHandler-Objekt.
createInventory()	Methode, die das Inventar erstellt.
createDeviceObject	Methode, die aus dem FileSystem die einzelnen Devices ausliest und für jedes ein Device-Objekt erstellt.
createDeviceConnectionObject	Methode, die aus dem FileSystem die Geräteverbindungen ausliest und für jede Verbindung ein deviceConnection-Objekt erstellt.

Connection

Die Connection-Klasse spezifiziert die Netzwerk-Schnittstelle, die für die Verbindung zwischen dem Programm und des zu Testenden Netzwerks verwendet werden soll. Für die Auswahl und Instanzierung wird eine ConnectionFactory verwendet und die Verbindungen lassen sich mittels einem Strategy-Pattern auswählen.

Komponenten	Beschreibung
connection	Ausgewählte Netzwerkschnittstelle, die dem TestBuilder als Rückgabewert geliefert wird.
logger	Referenz auf das Logger-Objekt.
connectionFactory	Referenz auf das ConnectionFactory-Objekt.
connectionContext	Referenz auf das ConnectionContext-Objekt.
createConnection()	Methode, die aufgrund der Parameter der Devices eine mögliche Netzwerkschnittstelle auswählt.

ConnectionContext

Anwendung des Strategy Pattern für die Netzwerkschnittstelle. Der ConnectionContext hält eine Referenz auf das konkrete Schnittstellen-Objekt.

ConnectionFactory

Anwendung des Factory-Pattern auf die Netzwerkschnittstelle. Die Factory ist zuständig für die Instanzierung der konkreten Schnittstelle, mit der das Programm die Netzwerkumgebung testet.

Logger

Der Logger wird verwendet, um wichtiche Informationen zentral zu speichern. Diese Informationen betreffen nur den Systemzustand des zu entwickelnden Systems. Der Logger speichert Fehlermeldungen, Erfolgsbenachrichtigungen und weitere Informationen, die es einem Entwickler erlauben, unerwartetes Verhalten der Software besser zu verstehen.

3.3 Systemsequenzdiagramme

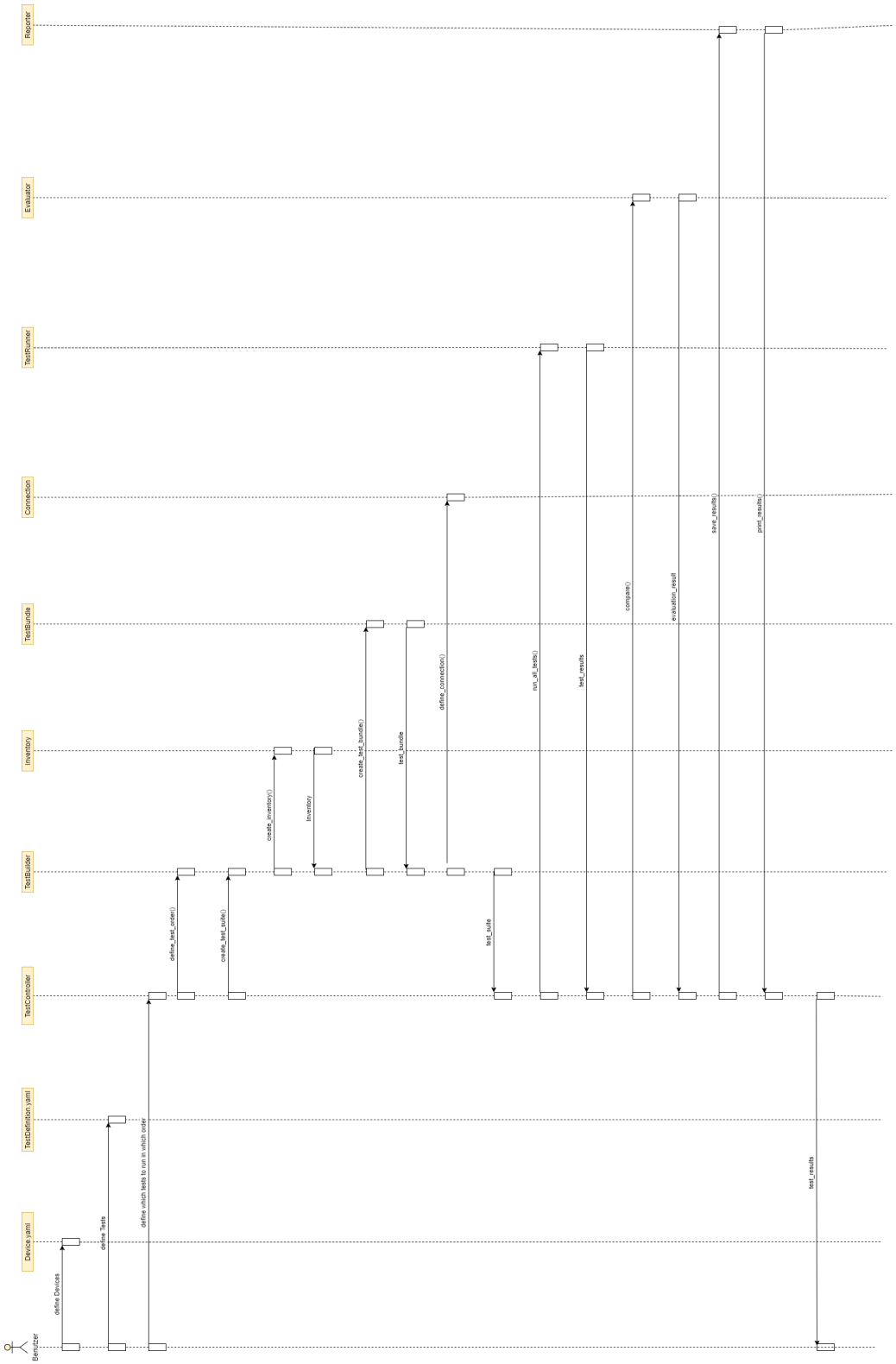

Abbildung 7: Systemsequenzdiagramm

3.3.1 TestBundle

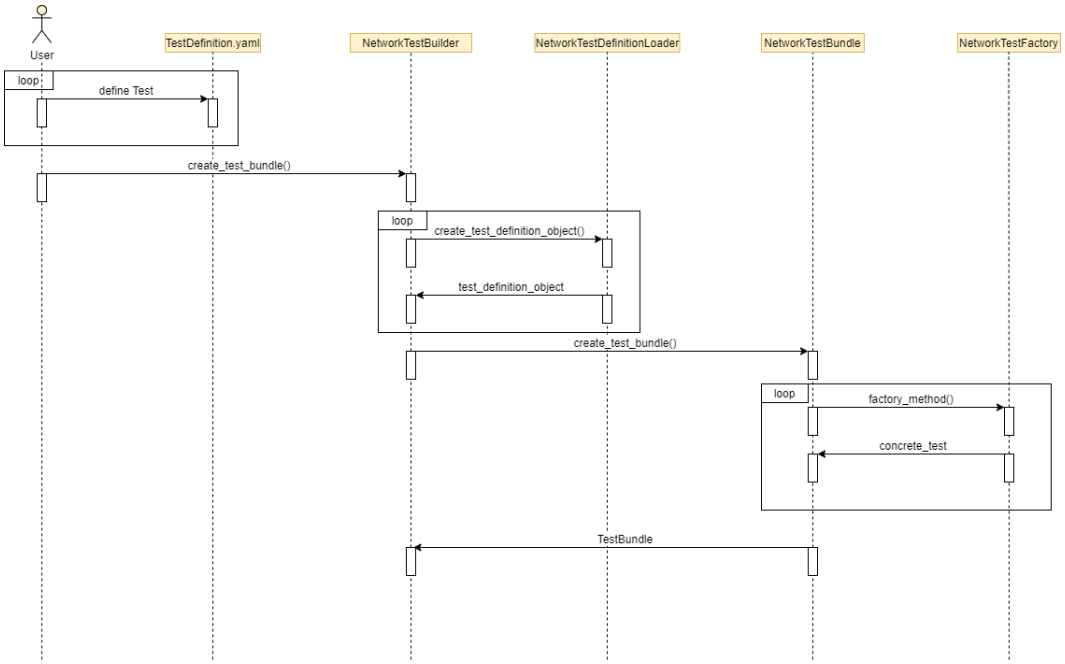

Abbildung 8: TestBundle Sequenzdiagramm

3.3.2 Inventar

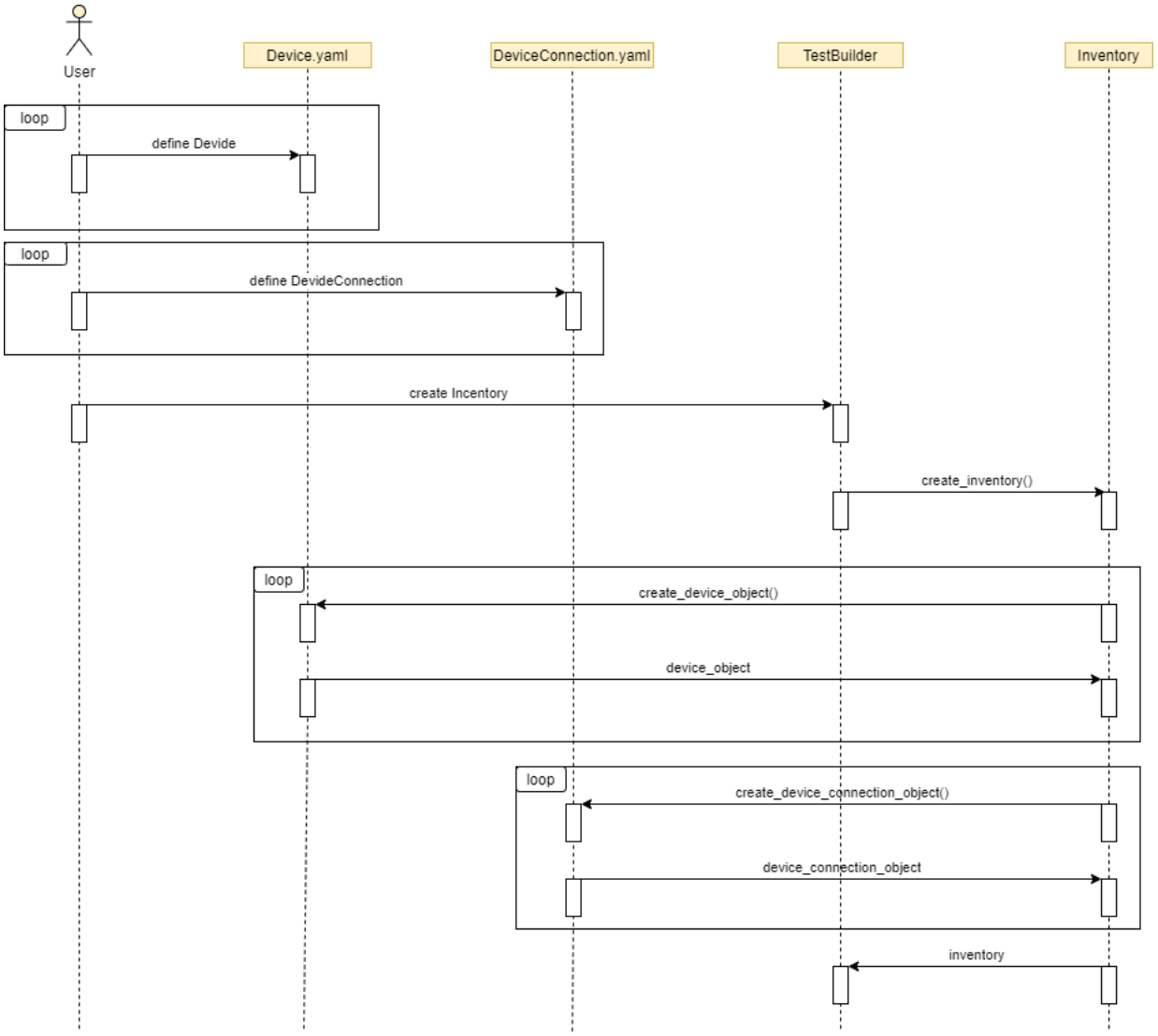

Abbildung 9: Inventar Sequenzdiagramm

3.3.3 Connection

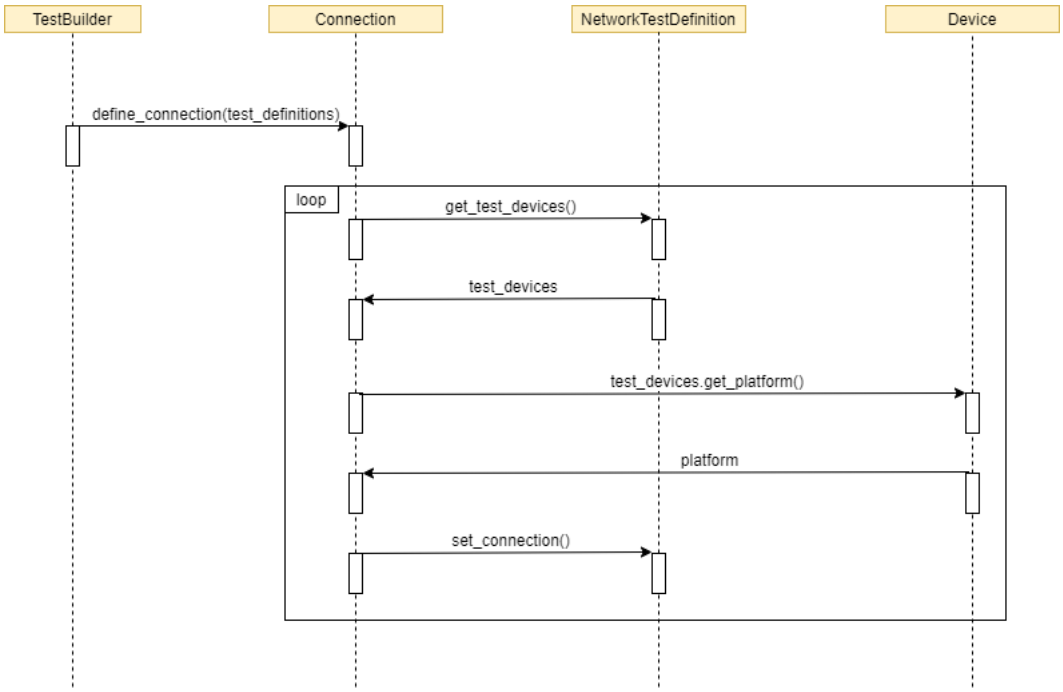

Abbildung 10: Connection Sequenzdiagramm

4 Architektur

4.1 Architekturentscheidungen

4.1.1 Anwendung von Patterns

Patterns in der Softwareentwicklung sind Vorschläge, wie mit bekannten Problemen umgegangen werden soll. Verschiedene Patterns können auf unterschiedliche Probleme angewandt werden und helfen, verschiedene Probleme zu lösen. Es gibt eine Vielfalt von Patterns mit vielen Anwendungsmöglichkeiten und Kombinationen. Für das Zu entwickelnde System wurden zwei Patterns ausgewählt, die die Erweiterbarkeit gewährleisten sollen.

Entscheidung 1: Anwendung von Patterns für die Testerstellung

- Im Kontext der Erweiterbarkeit des zu entwickelnden Systems,
- damit weitere Netzwerktest einfach hinzugefügt werden können,
- wurde entschieden, dass für die Testerstellung das Strategy und Factory-Method Pattern angewandt wird,
- um die Instanzierungslogik von den einzelnen Testklassen zu entkoppeln.
- Dabei wird akzeptiert, dass die Implementation des Systems alles in allem komplexer, und die Erstellung umfangreicher wird.

Das Strategy Pattern ist ein Verhaltenspattern und ermöglicht eine erhöhte Austauschbarkeit einzelner Softwarekomponenten. Eine gruppe von Algorithmen wird zu einer Strategie zusammengefasst und können untereinander leicht ausgetauscht werden.

Das Factory Method Pattern erlaubt die instanzierung von Objekten, ohne die spezifische Klasse des Objekts kennen zu müssen. Somit wird die Flexibilität der Software und die Austauschbarkeit der einzelnen Komponenten erhöht.

Weitere Software-Pattern werden im Verlauf der Arbeit bei Bedarf verwendet, sind aber nicht Teil der Architekturentscheidungen.

4.1.2 Testdefinitionssprache

Die Testdefinitionen benötigen ein einheitliches Format, damit sie vom Programm interpretiert werden können und dass daraus Tests erstellt werden können. Für diese Aufgabe gibt es eine Vielzahl von verschiedenen Beschreibungssprachen.

XML (Extensible Markup Language) ist eine Auszeichnungssprache, die in der Informatik oft angewandt wird, wenn Daten in einer Hierarchischen Struktur in einer Textdatei beschrieben werden. Der grösste Vorteil in der Verwendung von XML ist die hohe Verbreitung und Platformunabhängigkeit. Nachteile sind eine höhere Redundanz als in anderen Formaten was zu höherem Speicherverbrauch und grösserem Bandbreitenverwendung führt, wenn Daten im XML-Format gespeichert oder über das Internet versendet werden.

JSON (Javascript Object Notation) ist Menschenlesbar, Leichtgewichtig und wird zum effizienten Austausch von Informationen in einem geordneten Format auszutauschen. Das Format baut auf einer Attribut : Attributwert Logik auf und ist im Vergleich zu XML effizienter über das Internet austauschbar. Im Gegensatz zu XML bietet JSON aber keine Schemas, ein einheitliches Format für den Datenaustausch, was die Verwendung in verschiedenen Systemen komplizierter macht.

YAML (YAML ain't Markup Language) ist ein Datenserialisierungsformat mit Fokus auf Menschenlesbarkeit und wird hauptsächlich für die Erstellung von Konfigurationsdaten verwendet. Die grössten Vorteile sind die Datenkonsistenz der Datenmodel und die hohe Portierbarkeit zwischen verschiedensten Programmiersprachen. Zu den Nachteilen gehört die geringere Performanz gegenüber XML und JSON und die geringere Verbreitung von YAML. Ein weiterer Wichtiger Punkt ist, dass Netzwerktechniker in der Lage sein sollten, YAML zu lesen und schreiben.

Entscheidung 2: Auswahl der Testdefinitionssprache

- Um die Testdefinitionen in einer möglichst menschenlesbaren Form zu halten,
- und die Verwendbarkeit im zu entwickelnden System zu gewährleisten,
- haben wir entschieden, die Testdefinitionen in YAML zu verfassen,
- um ein Format zu verwenden, dass auch von Netzwerkleuten verwendet wird.
- Andere Technologien, wie JSON, XML usw. werden dabei voraussichtlich weggelassen,
- auch wenn diese in spezifischen Anwendungen wie die Speicherung von Daten geeignet wären.

4.1.3 Benutzeroberfläche

Damit Benutzer des Systems mit diesem interagieren können, wird eine Benutzeroberfläche benötigt. Das Design der Oberfläche spielt eine entscheidende Rolle für die Benutzbarkeit der Software. Mögliche Umsetzungen sind eine Grafische Benutzeroberfläche (GUI), die es Benutzern erlaubt über eine vordefinierte Oberfläche mit dem Programm zu interagieren. Weiterhin lässt sich das System auch mit einem Kommandozeilen-Userinterface umsetzen, welches weitaus einfacher in der Implementation ist, aber gegenüber dem GUI weniger Interaktionsmöglichkeiten bietet.

Entscheidung 3: Weglassen einer Grafischen Benutzeroberfläche

- Im Rahmen der zeitlichen Beschränkung der Arbeit,
- um uns auf die Implementation der Logik zu konzentrieren,
- werden wir kein GUI entwickeln.
- Stattdessen wird, wo nötig, eine Kommandozeileninteraktion implementiert.
- Es wird akzeptiert, dass die Benutzerfreundlichkeit des zu entwickelnden Systems dadurch eingeschränkt wird.

4.1.4 Testframework

Für die Kommunikation und das automatisierte Testen von Netzwerken gibt es verschiedene Möglichkeiten. Nachfolgend sind einige davon aufgelistet:

Ansible ist ein Framework für automatisierte Netzwerkoperationen wie Konfiguration, Skalierung oder Testing. Das Framework arbeitet, ohne dass auf dem Zielgerät etwas installiert werden muss, das mit der Software kommuniziert, solange das Zielgerät über SSH erreichbar ist. Ansible ist vergleichsweise einfach zu erlernen und in Python geschrieben. Die Netzwerkkonfiguration wird im YAML Format erfasst und das Programm sorgt dafür, dass die gewünschte Konfiguration im Netzwerk umgesetzt wird. Nachteile sind der geringe Windows-Support und das fehlende Auflösen von Abhängigkeiten.

Saltstack ist ein Konfigurations-Management-Framework mit Funktionen für automatische Neztwerktests. Parallele Programmausführung, Skalierbarkeit und eine aktive Community gehören zu den grössten Vorteilen. Allerdings ist der Installationsprozess für Neueinsteiger eher komplex und ausser für Linux ist der Betriebssystemsupport limitiert. Auch Saltstack ist in Python geschrieben, was die Erweiterbarkeit erhöht.

Puppet ist eine konfigurationsautomatisierungs- und Deployment-organisations-Lösung für verteilte Software und Infrastruktur. Geschrieben in Ruby, bietet Puppet eine Vielzahl von Funktionen für die Orchestrierung, Konfigurationsautomatisierung, Visualisierung und Reporting. Stärken sind der Support für eine Vielzahl von Betriebssystemen, hohe Stabilität und Robustheit. Schwierigkeiten können die hohe Anfängerschwierigkeit, schwächere Skalierbarkeit und Flexibilität (verglichen mit den anderen Lösungen).

Chef war ursprünglich ein Tool für internes End-zu-End Server Deployment und wurde später als Open Source Lösung für Cloud und Infrastruktur Automatisierung veröffentlicht. Es wird als eine der flexibelsten Lösungen für OS und Middleware Management beschrieben und wurde für Programmierer entwickelt. Chef hat eine ausführliche Dokumentation und ist sehr stabil und verlässlich für grosse Systeme. Die grössten Nachteile sind die steile Lernkurve und ein komplizierteres Inbetriebnehmen gegenüber den anderen Systemen.

Nornir wurde in der Studienarbeit von Anfang an als mögliche Lösung für die Testautomation genannt. Das Framework wurde in Python für die Verwendung mit Python entwickelt und bietet eine Vielzahl von Möglichkeiten für die Ausführung von Tests auf verschiedensten Systemen. Nornir führt die Tests als serielle oder parallele Tasks aus und verwendet YAML für die Datenverwaltung. Allerdings lässt sich Nornir nicht einfach ausführen, sondern muss in ein Python-Programms implementiert werden. Deshalb kann Nornir nicht einfach Installiert und danach verwendet werden, sondern ist ein Tool, welches es erlaubt, eigene Netzwerktasks zu definieren und auszuführen.

Entscheidung 4: Verwendung von Nornir als Testframework

- Für die Kommunikation mit verschiedenen Netzwerken,
- um die Implementation möglichst zu vereinfachen,
- wurde entschieden, dass das System das Nornir-Automations-Framework verwendet wird,
- weil Nornir dafür gedacht ist, in einem Python-Programm eingesetzt zu werden
- und eine umfangreiche Sammlung von nützlichen Funktionen bietet, mit denen man Netzwerktests ausführen kann.

Die gesamte Arbeit wurde von Anfang an darauf ausgelegt, mit Nornir als Automatisierunsframework zu arbeiten. Die Vielseitigen Methoden, um mit Netzwerkgeräten zu kommunizieren und die Vielzahl von möglichen Befehlen erlauben es, einfachen und effizienten Code zu schreiben, welcher die Netzwerke testet.

4.1.5 Datenhaltung

Für die Verwaltung von Daten, insbesondere das persistente Speichern und Laden, gibt es diverse Möglichkeiten. Da die Testdefinitionen in YAML verfasst werden, wäre für die Verwaltung der YAML-Files ein0e Dokumentdatenbank geeignet. Dokumentdatenbanken entspringen aus dem NoSQL Paradigma und haben kein fixes definiertes Schema, was eine flexiblere Verwendung ermöglicht, aber auch Fehleranfälliger ist. Üblicherweise werden Dokumentdatenbanken in einem Key-Value Format gehalten, was die Verwendung mit YAML vereinfacht.

Weiterhin liesse sich eine Relationale Datenbank implementieren, wobei dafür ein Datenbankschema entworfen werden müsste. Die Verwendung einer SQL-Datenbank würde zu einer stärkeren Konsistenz der Daten führen, da die Datenbank eine strenge Konsistenz erzwingt. Allerdings sind relationale Datenbanken im Vergleich zu NoSQL Datenbanken komplexer und lassen sich schwieriger Skalieren.

Eine weitere Möglichkeit für die Speicherung der Daten wäre eine Graphdatenbank. Graphdatenbanken speichern nicht nur die Daten in Form von Knoten, sondern auch die Beziehungen zwischen den Daten in Form von Kanten zwischen den Knoten. Graphalgorithmen erlauben es, sehr effiziente Abfragen auf die Daten auszuführen und Verknüpfungen zwischen den Daten abzufragen, für die eine Relationale Datenbank viel komplexere und langsamere Queries verwenden würde.

Ausserdem besteht noch die Möglichkeit, keine Datenbank zu implementieren und die Daten auf dem Clientsystem zu verwalten. Hierbei werden die YAML-Dokumente direkt auf der Dateiablage des Clients gespeichert. Der Ansatz ist definitiv simpler als die Verwendung einer Datenbank, allerdings kann dies dazu führen, dass Konsistenzprobleme auftreten. Die Anwendung ohne Datenbank legt somit mehr Verantwortung in die Hände der Benutzer, welche sich selber darum kümmern müssen, dass die Daten in der richtigen Form persistent gespeichert werden. Es ist trotzdem möglich, gemeinsam einen Datensatz zu halten, beispielsweise liesse sich die Testkonfiguration über ein Git-Repository verwalten und von verschiedenen Standorten her verwenden. Auch die Verwendung mit einem zentralen Filesystem, z.B. ein Ordner auf dem Server ist grundsätzlich möglich, erfordert aber geringfügige Anpassungen am Programm, damit die Pfade korrekt verwendet werden.

Entscheidung 5: Verzicht auf eine Datenbank

- Um Daten persistent zu speichern,
- damit sie vom System wieder abgerufen werden können,
- werden die Daten im YAML-Format abgelegt,
- und nicht in einer Datenbank,
- wobei ein Datenbanksystem für die zentralisierte Speicherung möglicherweise besser geeignet wäre.

4.1.6 Deployment

Die fertige Software muss auf verschiedenen Systemen mit unterschiedlichen Betriebssystemen einsetzbar sein. Die Installation soll dabei so einfach wie möglich sein und eine geringe Anzahl Abhängigkeiten an weitere Software haben. Wenn das Programm beispielsweise fünf weitere Programme benötigt, die alle parallel dazu installiert werden müssen, könnte das für Netzwerktechniker ein Grund sein, eine andere Software zu wählen. Ausserdem führt eine kompliziertere Installation dazu, dass der Einstieg für Unerfahrene Netzwerkleute schwieriger wird. Somit sollte die Auslieferung möglichst simpel erfolgen und geringe/keine Abhängigkeiten haben.

Eine Auslieferung als ausführbare Datei, z.B. im .Exe-Format für Windows-Benutzer, wäre sicher für die Benutzung am einfachsten, allerdings wäre das Programm dann nicht so einfach erweiterbar, weil für jede Änderung das Programm nochmals aktualisiert werden müsste.

Das Deployment als Dockercontainer ist, unabhängig der eigentlichen Auslieferung, auch eine mögliche Option. Diese Möglichkeit hat aber mehr Abhängigkeiten gegenüber einer einfachen ausführbaren Datei, da auf dem Client-Computer zuerst noch Docker installiert werden muss.

Die einfachste Variante ist die Auslieferung als reines Python-Skript. Der grösste Vorteil ist, dass sich ein Python Skript auf jedem System ausführen lässt, welches Python installiert hat. Die meisten Betriebssysteme haben heute per Default Python vorinstalliert, zumindest auf MacOS und Linux-Systemen. Somit liesse sich die Software, ohne weitere Software (ausser evtl Python) installieren zu müssen, verwenden.

Entscheidung 6: Deployment als Python-Skript

- Im Kontext des Deployments, der Auslieferung der Software,
- um das Programm auf möglichst vielen Plattformen einfach einsetzen zu können,
- wird das System in Form eines Python-Scripts ausgeliefert,
- und nicht als ausführbare Datei im .exe Format,
- da Python auf den meisten Linux-Basierten Systemen und einigen modernen Windows-Systemen bereits vorinstalliert ist,
- womit sich die Installation/Ausführung des Programms stark vereinfacht.
- Die Folge, dass auf einigen Geräten für die Programmausführung zuerst Python installiert werden muss, wird akzeptiert.

Eine Auslieferung per Dockercontainer oder über einen Server ist hierbei immer noch möglich. Es werden aber jeweils systemspezifisch Änderungen am Programm/Konfiguration benötigt.

4.1.7 Security

Sicherheitspraktiken schlagen vor, dass Passwörter nur gespeichert werden sollen, wenn die Verwendung eines Systems eine Authentifizierung benötigt. Passwörter sollten in dem Falle mit einem Hash und Salt encoded und in dieser Form gespeichert werden. Das zu entwickelnde System benötigt Passwörter, um mit den Netzwerkgeräten kommunizieren zu können. Jedes Gerät benötigt (üblicherweise) einen Login-Namen und ein Passwort, um darauf die Konfiguration anzusehen, zu verändern oder Befehle auf dem Netzwerkgerät ausführen zu können.

Es wird davon abgeraten, eigene Algorithmen für die Verschlüsselung von Daten zu implementieren, da diese nicht getestet sind. Stattdessen sollen Bibliotheken verwendet werden, die von Kryptographen und Sicherheitsexperten getestet und als sicher eingestuft wurden.

Ein Nachteil für die Verwendung von Sicherheitspraktiken ist die geringere Benutzbarkeit. Es ist nicht möglich, nachzusehen, ob das Passwort, welches für ein Netzwerkgerät gesetzt wurde, immer noch korrekt ist, da das Hashing dies unmöglich macht. Deshalb ist es schwierig, das Programm so zu entwickeln, dass es sich sicher und simpel verwenden lässt, ohne dafür erheblichen Mehraufwand zu betreiben.

Entscheidung 7: Umgang mit Passwörtern

- Im Hinblick auf die Sicherheit des Systems,
- um die Verwendung des Programms möglichst einfach zu halten,
- werden Passwörter für die verschiedenen Netzwerkgeräte als Plaintext gespeichert,
- und nicht in verschlüsselter Form.
- Der sichere Umgang mit den Konfigurationsdaten ist somit in der Verantwortung des Benutzers.

Die Sichere Verwendung der Software würde voraussetzen, dass sich der Benutzer gegenüber des zu entwickelnden Programms authentifizieren muss. Nachdem der User verifiziert wurde, werden die bis dahin verschlüsselten Konfigurationsdaten entschlüsselt und lassen sich bearbeiten/benutzen. Eine weitere Möglichkeit wäre die Verwendung einer Datenbank, um die Konfigurationsdaten zu verwalten. Somit würde die Verantwortung für die sichere Speicherung bei der Datenbank und nicht beim Testprogramm liegen.

Beide Optionen werden als mögliche Erweiterung des Systems in Betracht gezogen.

4.2 Systemübersicht

Die Systemübersicht gibt einen Überblick über die verschiedenen Komponenten des Systems. Nachfolgend sind die einzelnen Komponenten detaillierter beschrieben.

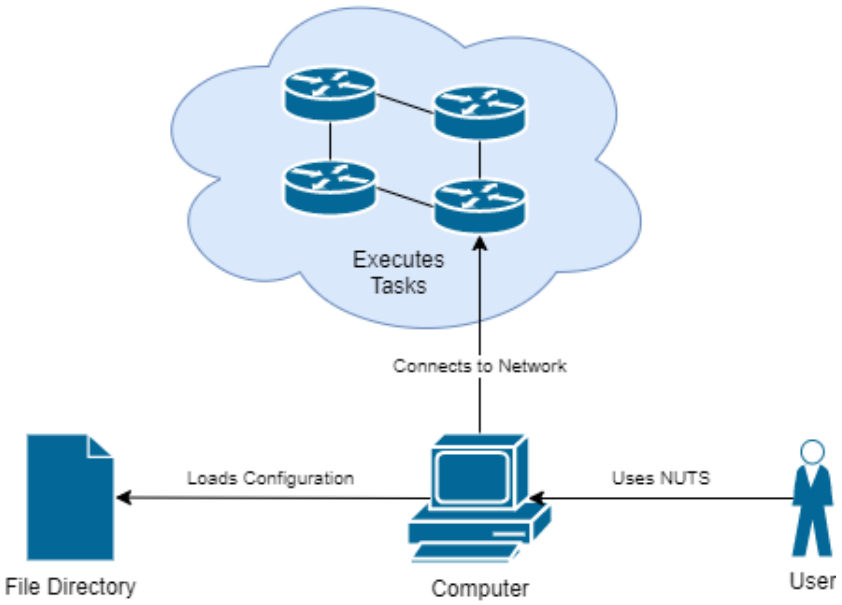

Abbildung 11: Systemübersicht

Computer

Dies entspricht unserem Client. Auf dem Computer läuft das zu entwickelnde System, welches aus einem Python Programm besteht und mittels Nornir mit einem Netzwerk kommuniziert und Tests darauf ausführt.

Netzwerk

Dies ist das Netzwerk, auf dem die automatisierten Tests ausgeführt werden sollen. Es besteht aus mehreren Netzwerkknoten, z.B. Switches, Router und Clients. Das Netzwerk wird über eine lokale Schnittstelle oder über das Internet angesteuert.

Datenablage

Die Datenablage befindet sich auf dem Computer und kann bei Bedarf über ein Verwaltungstool mit anderen Clients geteilt werden. Darin befinden sich sämtliche für das Testsystem benötigte Daten. Diese Daten werden in Form von YAML-Files als Key-Value Store angelegt.

4.3 Deployment

4.3.1 Deploymentdiagramm

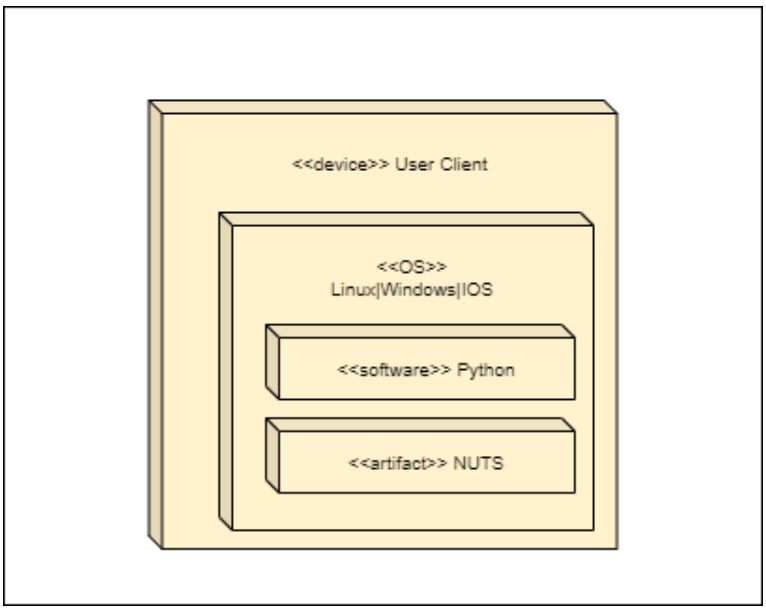

Abbildung 12: Deploymentdiagramm

Client

Der User Client kann ein Computer oder ein Server sein, welcher die automatisierten Tests ausführen soll. Darauf läuft ein beliebiges Betriebssystem

Betriebssystem

Es gibt keine spezifischen Anforderungen an das Betriebssystem, welches auf dem Client installiert ist. Python Code lässt sich auf beliebigen Betriebssystemen ausführen.

Python Installation

Für die Ausführung von Python-Code muss Python auf dem Client installiert sein.

NUTS2.0

Das zu entwickelnde Programm, welches auf dem Client installiert wird. Es verbindet sich mit einem Netzwerk und führt vorher definierte Tests darauf aus.

5 Zeitauswertung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6 Protokolle

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

7 Be(nuts)eranleitung

7.1 Installation

7.1.1 Installationsvoraussetzungen

NUTS2.0 sollte sich auf jedem Betriebssystem ausführen lassen. Für die Ausführung wird eine Python 3.7 installation (oder aktueller) benötigt. Diese kann unter https://www.python.org/downloads/ heruntergeladen werden.

Um NUTS2.0 ausführen zu können, muss zuerst das Github-Repository geklont werden: https://github.com/EkoGuandor229/Network-Unit-Testing. Danach können die verwendeten Module über das Requirements-File mit dem Befehl 'pip install requirements.txt' installiert werden.

7.1.2 Ausführen

Das Programm kann zu Testzwecken regulär in einer Programmierumgebung wie zum Beispiel PyCharm ausgeführt werden.

Um NUTS2.0 aus einer Konsole zu starten muss zum Root-Ordner NUTS2.0 navigiert werden. Z.B. C:Programs/Python/NUTS2.0.

Man kann das Programm mit dem Befehl: 'python -m nuts' starten. Wenn man das GUI auslassen und direkt alle Tests ausführen möchte kann man mit dem Befehl: 'python -m nuts -r' starten.

7.1.3 Konfiguration

Im File 'Config.yaml' können die Pfäde aller Ordner geändert werden, um beispielsweise das Inventory oder die Resultate zentral in einem Repository zu verwalten. Zusätzlich kann noch bestimmt werden, ob das GUI Per default übersprungen werden soll.

Attribut	Beschreibung
device_id	Eine eindeutige ID für das Device
platform	Das OS welches das Device benutzt
username	Der username welches das Device für das Login benutzt
password	Das Passwort welches das Device für das Login benutzt
hostname	Die Ip Adresse über welche das Device angesprochen werden kann
loopback-addresse	IP-Addresse des Loopback-Interface. Für einige Tests benötigt.

Tabelle 12: Deviceparameter

7.2 Inventar

Um Netzwerktests auszuführen benötigt man zuerst ein Inventar mit Devices und Device Connections. Die Devices sind die Netzwerkgeräte wie zum Beispiel Router oder Switches. Die Device Connections sind die Verbindungen zwischen den Devices.

7.2.1 Devices

Die Definitionen der Devices sind unter Resources/Inventory/Devices/Devices.yaml abgelegt:

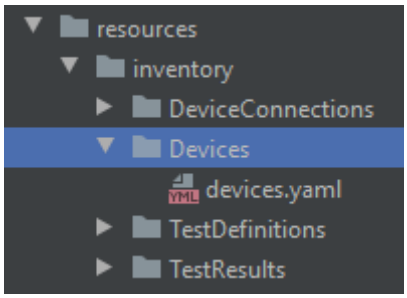

Abbildung 13: Position devices.yaml

Um neue Devices zu erfassen müssen folgende Informationen im yaml eingegeben werden:

Diese Informationen sollten gemäss folgendem Beispiel dargestellt werden:

7.2.2 Device Connections

Die Definitionen der Device Connections sind unter Resources/Inventory/DeviceConnections/Device-Connections.yaml abgelegt

Um Device Connections zu erfassen müssen folgende Informationen im yaml eingegeben werden:

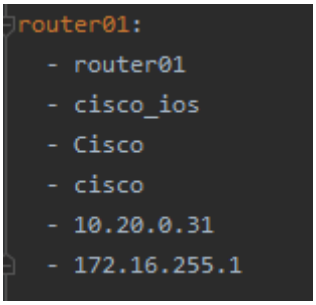

Abbildung 14: Beispiel-Erfassung Device

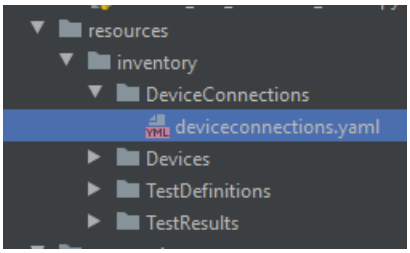

Abbildung 15: Position deviceconnections.yaml

Diese Informationen sollten wie folgt dargestellt werden:

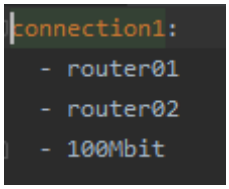

Abbildung 16: Beispiel-Erfassung Connection

7.3 Netzwerktests

Die Netzwerktests sind die Tests, welche effektiv auf dem zu testenden Netzwerk ausgeführt werden sollen. Die Testdefinitionen befinden sich unter Resources/Inventory/TestDefinitions:

Attribut	Beschreibung
device a	Die ID des ersten Devices
device b	Die ID des zweiten Devices
connection speed	Die Übertragungsrate der Verbindung

Tabelle 13: DeviceConnection Parameter

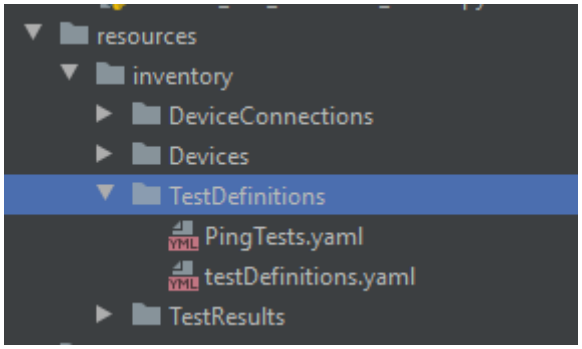

Abbildung 17: Position Testdefinitionen

Attribut	Beschreibung
test_id	Eine eindeutige ID für den Test
command	Ein Command um den Test zu bestimmen
test_device	Die ID des Devices auf welchem der Test ausgeführt werden soll
target	Das Ziel (Zum Beispiel eine IP im Falle eines Pings)
expected_result	Das erwartete Resultat (Zum Beispiel Success im Falle eines Pings)
test_group	Ein Gruppenname um später die Tests zu Kategorisieren

Tabelle 14: Testdefinition Parameter

Es können in diesem Ordner beliebig viele YAML Files abgelegt werden und es werden vonn allen Files die Tests erfasst.

Um die Tests zu erfassen müssen folgende Informationen im yaml eingegeben werden:

Diese Informationen sollten wie folgt dargestellt werden:

```
test01:
  - PingRouter1Loopback0
  - Ping
  - router01
  - 172.16.255.1
  - Success
  - connectivity
```

Abbildung 18: Beispiel-Erfassung Testdefinition

7.3.1 Commands

Folgende Commands sind bereits implementiert und können mit den jeweiligen expected_results verwendet werden:

Ping Führt einen Ping-Test auf das spezifizierte Target mit 4 ICMP Packeten aus. In der jetzigen Konfiguration wird eine 100% Erfolgsquote erwartet, damit der Ping-Test als erfolgreich gilt. Wenn andere Werte erwartet werden, muss dafür ein neuer Ping-Test implementiert werden, welche unterschiedliche Erwartungswerte implementiert hat.

Als erwartetes Resultat kann 'Success' oder 'Failure' verwendet werden.

Show Interfaces Der 'Show Interfaces'-Befehl benötigt kein Target, da die Interfaces des test_device abgefragt werden. Bei der Definition kann somit einfach 'No Target' eingegeben werden. Als erwartetes Resultat wird ein Dictionary mit key:'Interfacename' und value:'True' oder 'False' verwendet werden. Dies sollte in folgender Form dargestellt werden:

```
- {
    'GigabitEthernet1': True,
    'GigabitEthernet2': True,
    'GigabitEthernet3': True,
    'GigabitEthernet4': True,
    'Loopback0': True
  }
```

Abbildung 19: Show Interfaces Erwartungswert

Traceroute Führt einen Traceroute vom test_device auf das in target angegebene Ziel aus. Als erwartetes Resultat wird ein Array von IP Adressen angegeben. Diese müssen in der Reihenfolge, in denen die Hops im Traceroute besucht werden, angegeben werden. Für Hops, die keine IP-Addresse anzeigen, kann ein '*' in das Array eingetragen werden.

Das Array bei expected_result kann beispielsweise so aussehen:

```
- ["172.16.13.1", "172.16.14.4"]
```

Abbildung 20: Traceroute Erwartungswert

Parametername	Parameterwert
'interface':	Name des Interface.
'mac':	MAC-Addresse des Nachbargeräts
'ip':	IP-Addresse des Nachbargeräts

Tabelle 15: ARP Table Erwartungswert Parameter

Parametername	Parameterwert
'Neighbor-ID':	IP-Addresse des Nachbargeräts
'Priority':	OSPF-Priorität als Ganzzahl
'State':	Status des Nachbargeräts
'Address':	IP-Addresse des Interface, über welches der Nachbar erreichbar ist.
'Interface':	Name des Interface, über welches der Nachbar erreichbar ist.

Tabelle 16: OSPF Neighbor Erwartungswert Parameter

Arp Table Für den Befehl 'Arp Table' wird kein Target benötigt, da der Arp Table des im test_device angegebenen Netzwerkgeräts abgefragt wird. Es kann 'No Target' im 'target'-Feld eingegeben werden. Als erwartetes Resultat wird ein Array von Dictionaries erwartet. In den Dictionaries werden folgende Informationen erwartet:

Im Bild ist ein Beispiel des Arrays mit den Dictionaries in jeder Zeile angegeben:

```
- [
    {'interface': 'GigabitEthernet3', 'mac': '52:54:00:67:A5:39', 'ip': '172.16.12.1'},
    {'interface': 'GigabitEthernet3', 'mac': '52:54:00:28:F0:9D', 'ip': '172.16.12.2'},
    {'interface': 'GigabitEthernet4', 'mac': '52:54:00:16:91:60', 'ip': '172.16.13.1'},
    {'interface': 'GigabitEthernet4', 'mac': '52:54:00:12:E0:4C', 'ip': '172.16.13.3'},
    {'interface': 'GigabitEthernet2', 'mac': '52:54:00:63:CD:6C', 'ip': '172.16.14.1'},
    {'interface': 'GigabitEthernet2', 'mac': '52:54:00:5D:8E:C7', 'ip': '172.16.14.4'}
  ]
```

Abbildung 21: ARP Table Erwartungswert

Ospf Neighbor Für den Befehl 'Ospf Neighbor' wird kein Target benötigt, da OSPF Nachbarn des im test_device angegebenen Netzwerkgeräts abgefragt wird. Es kann 'No Target' im 'target'-Feld eingegeben werden. Als erwartetes Resultat wird ein Array mit Dictionaries erwartet. In den Dictionaries werden folgende Informationen benötigt:

Das folgende Beispiel zeigt das Array von Dictionaries, wie es im YAML dargestellt wird:

Abbildung 22: OSPF Neighbor Erwartungswert

7.4 Durchführung

Nachdem dass Inventar erstellt und die Testdefinitionen erfasst wurden, kann man das Programm starten. Falls die Option Skip-GUI aktiviert wurde, werden alle Tests in der Reihenfolge durchgeführt, in der sie in der Testdefinition angegeben wurden. Falls dies nicht aktiviert wurde öffnet sich ein Grafikinterface.

7.4.1 GUI

Das GUI für die Definition der Test Reihenfolge besteht aus zweit Tabs:

Im ersten Tab werden alle Tests nach Gruppen sortiert angezeigt. Die Gruppierung ist diejenige, die in der Testdefinition angegeben wurde. Hier kann der Benutzer auswählen welche Tests er ausführen möchte. Nachdem die Tests ausgewählt wurden weden mit dem Button 'Select' alle ausgewählten Tests selektiert und man kann danach die Reihenfolge der selektierten Test im zweiten Tab einstellen.

Im zweiten Tab werden alle selektierten Tests angezeigt und der Benutzer kann mit den jeweiligen Buttons die Reihenfolge bestimmen. Nachdem der Benutzer mit der Reihenfolge zufrieden ist, kann mit dem Button 'Save and Quit' das GUI beendet werden. Alle selektierten Tests werden nun in der angegebenen Reihenfolge durchgeführt.

7.4.2 Test Resultate

Die Resultate der jeweiligen Durchführungen werden in der Konsole angezeigt und zusätzlich noch in einem File gespeichert. Bestandene Tests werden nur mit ihrem Namen und dem Vermerk, dass der Test bestanden ist, angegeben. Nicht bestandene Tests haben den Testnamen, das erwartete Ergebniss und das tatsächliche Ergebniss für einen soll-ist-Vergleich.

Das folgende Bild zeigt eine komplette Durchführung des Programms mit elf bestandenen Tests und null nicht bestandenen Tests:

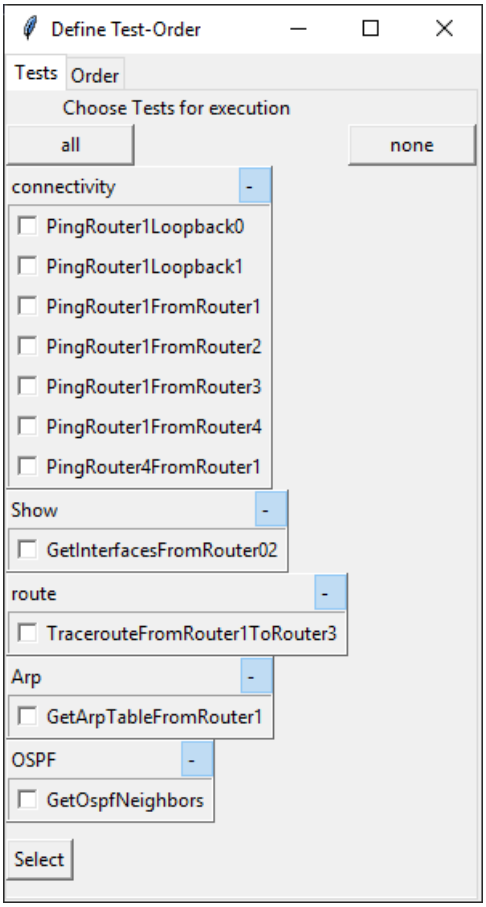

Abbildung 23: GUI Testauswahl

Das File, in dem der Testreport abgespeichert wird, befindet sich unter: 'Resources/Inventory/Test-Results/results.txt'.

In dem File wird zuerst der Zeitstempel der Testdurchführung angegeben, danach werden zuerst die bestandenen Tests und am Schluss die nicht bestandenen Tests angezeigt. Bei den nicht bestandenen Tests werden zusätzlich noch das erwartete und das tatsächliche Resultat angezeigt.

7.5 Neue Tests hinzufügen

Falls der Benutzer eigene Tests hinzufügen möchte, müssen an folgenden Orten änderungen vorgenommen werden:

7.5.1 Conctrete Tests

Die konkreten Tests befinden sich unter 'nuts/testcreation/concretetests' In diesem Ordner muss ein neues Python-File angelegt werden. Im File wird der Test als Klasse implementiert. Es ist darauf zu

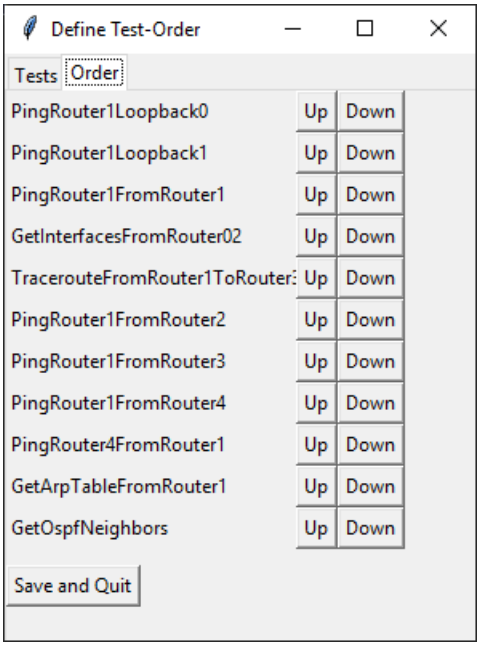

Abbildung 24: GUI Testreihenfolge

achten, dass das Basisinterface 'NetworkTestStrategyInterface' von der Testklasse implementiert wird, um davon die grundlegenden Funktionalitäten zu erben.

Nach der Erstellung muss der Test geschrieben werden. Dazu wird in der ___init___ Methode die Funktion self.nr. = InitNornir() aufgerufen und darin die benötigten Parameter übergeben.

In der Methode run_test(): wird angegeben, welches Nornir-Plugin mit welchem Task verwendet wird, z.B. task=napalm_get.

In der set_result() Methode wird die Logik für das Parsen des Rückgabewerts des Tests implementiert. Es ist darauf zu achten, dass dabei das Resultat in ein einheitliches Format gebracht wird, so dass man in der evaluate_result() Methode das erwartete Ergebnis möglichst mit einem == zum tatsächlichen Ergebnis vergleichen kann. Falls dies nicht möglich ist, muss im evaluate_result() zusätzlich Logik implementiert werden, um die Resultate zu vergleichen.

Mehr Informationen, welche Begehle mit Nornir ausgeführt werden können, findet man auf https://nornir.readthedocs.io

Informationen zum Napalm-Treiber findet man auf https://napalm.readthedocs.io

Die bereits erstellten Tests können als Vorlage für weitere Testimplementationen verwendet werden.

```
    _   ____  ____________    ___    ____
   / | / / / / /_  __/ ___/   |__ \  / __ \
  /  |/ / / / / / /  \__ \    __/ / / / / /
 / /|  / /_/ / / /  ___/ /   / __/_/ /_/ /
/_/ |_/\____/ /_/  /____/   /____(_)____/


+-----------------------------------------------------------------------------------------------------+
| INITIALIZING TEST SUITE                                                                             |
+-----------------------------------------------------------------------------------------------------+
 Create Device Objects from YAML: 100%|████████████████████████████| 4/4 [00:00<00:00, 4009.85it/s]
 Read objects from PingTests.yaml: 100%|████████████████████████████| 2/2 [00:00<00:00, 2004.93it/s]
 Read objects from testDefinitions.yaml: 100%|████████████████████████| 9/9 [00:00<00:00, 8934.61it/s]
 Create runnable tests: 100%|████████████████████████████████████| 11/11 [00:00<00:00, 479.54it/s]
+-----------------------------------------------------------------------------------------------------+
| RUN ALL TESTS                                                                                       |
+-----------------------------------------------------------------------------------------------------+
 Execute tests: 100%|████████████████████████████████████████████| 11/11 [01:30<00:00,  8.19s/it]
+-----------------------------------------------------------------------------------------------------+
| TEST RESULTS                                                                                        |
+-----------------------------------------------------------------------------------------------------+
Passed Tests
|   |-- Test: PingRouter1Loopback0 has PASSED
|   |-- Test: PingRouter1Loopback1 has PASSED
|   |-- Test: PingRouter1FromRouter1 has PASSED
|   |-- Test: GetInterfacesFromRouter02 has PASSED
|   |-- Test: TracerouteFromRouter1ToRouter3 has PASSED
|   |-- Test: PingRouter1FromRouter2 has PASSED
|   |-- Test: PingRouter1FromRouter3 has PASSED
|   |-- Test: PingRouter1FromRouter4 has PASSED
|   |-- Test: PingRouter4FromRouter1 has PASSED
|   |-- Test: GetArpTableFromRouter1 has PASSED
|   |-- Test: GetOspfNeighbors has PASSED
-------------------------------------------------------------------------------------------------------
No tests failed
```

Abbildung 25: Programmausführung im PyCharm

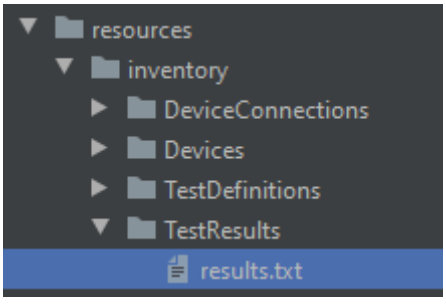

Abbildung 26: Position Testresultate

```
New Test Run on 2020-05-13 10:10:54.465216:
  Passed Tests:
    Test: PingRouter1Loopback0 has PASSED
    Test: PingRouter1Loopback1 has PASSED
    Test: PingRouter1FromRouter1 has PASSED
    Test: GetInterfacesFromRouter02 has PASSED
    Test: TracerouteFromRouter1ToRouter3 has PASSED
    Test: PingRouter1FromRouter2 has PASSED
    Test: PingRouter1FromRouter3 has PASSED
    Test: PingRouter1FromRouter4 has PASSED
    Test: PingRouter4FromRouter1 has PASSED

  Failed Tests:
    Test: GetArpTableFromRouter1 has FAILED
      Expected: [{'interface': 'GigabitEthernet3', 'mac': '52:54:00:67:A5:39', 'ip': '172.16.12.1'},
                 {'interface': 'GigabitEthernet3', 'mac': '52:54:00:28:F0:9D', 'ip': '172.16.12.2'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:16:91:60', 'ip': '172.16.13.1'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:12:E0:4C', 'ip': '172.16.13.3'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:63:CD:6C', 'ip': '172.16.14.1'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:5D:8E:C7', 'ip': '172.16.14.1'}]

      Actual:   [{'interface': 'GigabitEthernet3', 'mac': '52:54:00:67:A5:39', 'ip': '172.16.12.1'},
                 {'interface': 'GigabitEthernet3', 'mac': '52:54:00:28:F0:9D', 'ip': '172.16.12.2'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:16:91:60', 'ip': '172.16.13.1'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:12:E0:4C', 'ip': '172.16.13.3'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:63:CD:6C', 'ip': '172.16.14.1'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:5D:8E:C7', 'ip': '172.16.14.4'}]

End of Test Run
```

Abbildung 27: Testresultate.txt

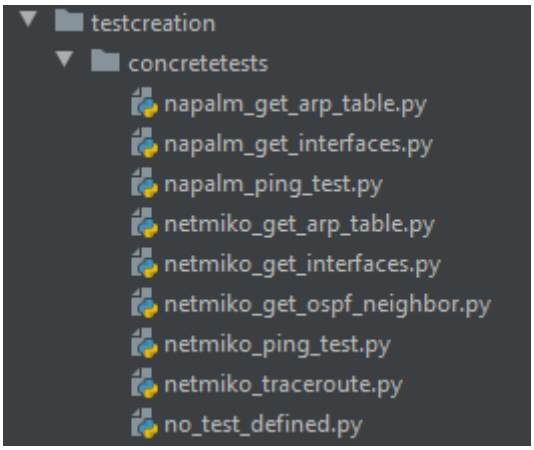

Abbildung 28: Position konkrete Netzwerktests

7.5.2 Network Test Strategy Factory

Die Network Test Strategy Factory implementiert die Logik, nach der die Tests ausgewählt und instanziert werden. Dafür werden sämtliche Tests in einer test_map gespeichert. Die test_map ist ein Dictionary von Dictionaries und hat als äusseren Key den Befehl, welcher Test ausgeführt werden soll und als inneren Key die Connection, die für den Test verwendet wird. Als Value ist die konkrete Klasse eingetragen, die für den Test instanziert werden soll. Gibt es für eine Kombination aus Command-Connection keinen konkreten Test, muss hier statt der Testklasse ein 'None' angegeben werden. Falls in der Instanzierungslogik ein Test, welcher in der Testdefinition angegeben wurde, nicht existiert, wird stattdessen ein NoTestDefined-Test instanziert, welcher in der Evaluation immer 'nicht bestanden' zurückgibt mit der Anmerkung, dass dieser Test noch nicht erstellt wurde.

Das folgende Bild zeigt die test_map mit den oben beschriebenen Werten.

```
def __init__(self):
    self.test_map = {
        "Ping": {
            "Napalm": NapalmPingTest,
            "Netmiko": NetmikoPingTest,
        },
        "Show Interfaces": {
            "Napalm": NapalmShowInterfaces,
            "Netmiko": NetmikoShowInterfaces
        },
        "Traceroute": {
            "Napalm": None,
            "Netmiko": NetmikoTraceroute
        },
        "Arp Table": {
            "Napalm": NapalmShowArpTables,
            "Netmiko": NetmikoShowArpTables
        },
        "Ospf Neighbor": {
            "Napalm": None,
            "Netmiko": NetmikoShowOspfNeighbor
        }

        # Add more Tests as "Testcommand: {connection_dictionary}
    }
```

Abbildung 29: TestMap in der TestStrategyFactory

Persönliche Berichte

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Persönlicher Bericht Mike Schmid

Zu Beginn der Studienarbeit war ich nicht sicher, was mich erwartet.

Eigenständigkeitserklärung

Die Authoren erklären hiermit,

- dass die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt wurde, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit den Betreuerm vereinbart wurde.
- dass sämtliche verwendeten Quellen erwähnt und gemäss den gängigen wissenschaftlichen Zitierregeln korrekt angegeben wurden.
- dass keine durch Uhrheberrecht geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter weise verwendet wurden.

Ort, Datum:	Ort, Datum:
Name, Unterschrift:	Name, Unterschrift:

Vereinbarung Urheber-/ Nutzungsrechte

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit NUTS2.0 von Janik Schlatter und Mike Schmid unter der betreuung von Beat Stettler und Urs Baumann geregelt

2. Urheberrecht

Die Urheberrechte stehen den Authoren zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von den Authoren, von der OST (ehemals HSR), sowie vom INS Institut for Networked Solutions nach Abschluss der Arbeit verwendet und weiter entwickelt werden.

4. Softwarelizenz

Die in der Arbeit entstandene Software untersteht der Apache Licence 2.0. Somit darf die Software frei verwendet, verändert und verteilt werden, insofern dies nicht die in der Lizenz genannten Regelungen verletzt.

Rapperswil, den..... .....
Die Studentin/ der Student

Rapperswil, den..... .....
Die Studentin/ der Student

Rapperswil, den..... .....
Der Betreuer/ die Betreuerin der Studienarbeit

Rapperswil, den..... .....
Der Betreuer/ die Betreuerin der Studienarbeit

Glossar

Begriffserklärung

TODO

Abbildungsverzeichnis

*

Tabellenverzeichnis

*

Literaturverzeichnis

TODO