Automatisierung der Berichterstellung für Konformitätstests von Zugsicherungskomponenten

Bachelorarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

Michelle Knop

Abgabe am 24. August 2022

Bearbeitungszeitraum:	07.06.22 – 24.08.22
Matrikelnummer, Kurs:	7448173, TINF19IT1
Ausbildungsbetrieb:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer des Ausbildungsbetriebs:	Dipl.-Ing. Lennart Asbach
Gutachter der Dualen Hochschule:	Prof. Dr. Nathan Sudermann-Merx

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem

Thema

Automatisierung der Berichterstellung für Konformitätstests von Zugsicherungskomponenten

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* falls beide Fassungen gefordert sind

Braunschweig, den 24. August 2022

Zusammenfassung

Für einen reibungslosen Schienenverkehr spielt die Zulassung der Zugsicherungskomponenten eine große Rolle. Hierfür werden Konformitätstests von unabhängigen Laboren durchgeführt. Eines dieser Labore befindet sich am Deutschen Zentrum für Luft- und Raumfahrt (DLR) in Braunschweig in dem Institut für Verkehrssystemtechnik. Hier werden Zugrechner geprüft und anschließend die Daten ausgewertet. Das Ergebnis eines solchen Konformitätstests ist ein Testbericht, welcher dem Kunden als ein Feedback dient.

Um die Erstellung von diesen Testberichten zu vereinfachen, wird im Rahmen dieser Arbeit die Automatisierung der Testberichterstellung umgesetzt. Das Ziel dieser Arbeit sind Berichte, welche regelmäßige und mit wenig Aufwand des Benutzers aktualisiert werden können.

Dazu wurde eine Konfigurationsdatei erstellt, welche alle Parameter enthält, die der Benutzer für die Erstellung anpassen muss. Außerdem werden durch ein Automatisierungsskript, welches regelmäßig durch Jenkins ausgeführt wird, Log-Daten, die Konfigurationsdatei und eine Datenbank auf Änderungen überprüft. Diese werden automatisch verarbeitet und in die neue Version des Berichts eingebunden. Durch dieses Vorgehen liegt dem Nutzer zu jeder Zeit ein aktualisierter Testbericht vor, in welchen alle aktuellen Daten eingebunden sind.

Abstract

The testing of railway control and safety systems plays a major role for a smooth rail traffic. For this purpose, independent laboratories conduct functional conformity tests. One of those laboratories is located at the German Aerospace Centre (DLR) in Braunschweig in the Institute of Transportation Systems. Here, **E**uropean **T**rain **C**ontrol **S**ystem (ETCS) on-board equipment is checked and the data is evaluated. The test result is summarized in a report. This serves as a feedback for the customer.

To facilitate the generation of the report, the automation of this process is implemented in the scope of this thesis. The aim of this thesis are reports which are regulary updated with minimal user effort.

In doing so, a configuration file is developed, containing all parameters that are integrated in the report. This file must be individually customized for every report. Moreover, an automation script is checking the log data, the configuration file and a database. The script is regularly executed by Jenkins. Changes in the data are automatically processed for a new version of the report. Hereby, the user has an updated test report, which contains all up-to-date data, at all times.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Anhangsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

BPA	**B**usiness **P**rocess **A**utomation
DAkkS	**D**eutsche **Akk**reditierungs**s**telle
DHBW	**D**uale **H**ochschule **B**aden-**W**ürttemberg
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt e. V.
DMI	**D**river **M**achine **I**nterface
EIU	**E**isenbahn**i**nfrastruktur**u**nternehmen
ERA	**E**uropean Union **A**gency for **R**ailways
EVC	**E**uropean **V**ital **C**omputer
ETCS	**E**uropean **T**rain **C**ontrol **S**ystem
JRU	**J**uridical **R**ecording **U**nit
KI	**K**ünstliche **I**ntelligenz
OBU	**O**n-**B**oard **U**nit
PDF	**P**ortable **D**ocument **F**ormat
RailSiTe	**Rail**way **Si**mulation and **Te**sting
TSEval	**T**est **S**equence **Eval**uation
TSI CCS	**T**echnical **S**pecification for **I**nteroperability relating to the subsystem "**C**ontrol-**C**ommand and **S**ignalling"
WHL	**Wh**eel

1. Einleitung

Für einen sicheren Schienenverkehr wurde das **E**uropean **T**rain **C**ontrol **S**ystem (ETCS)[1] entwickelt. Hierbei handelt es sich unter anderem um die Digitalisierung der Streckensignale und Lichtzeichen an den Gleisen. Die Informationen werden bei vollständig ausgestatteten Strecken und Zugkomponenten auf dem Monitor in einem Führerraum angezeigt, wodurch auf optische Streckensignale verzichtet werden kann. Für die Interoperabilität zwischen den verschiedenen **E**isenbahn**i**nfrastruktur**u**nternehmen (EIU) und an den Landesgrenzen und für einen reibungslosen Schienenverkehr wurde sich mit dem ETCS auf einen einheitlichen Standard geeinigt. Damit können die Züge auf allen Strecken fahren, wenn sie die Anforderungen der Streckenseite softwaretechnisch in dem Zug erfüllen. [3][6]

Die korrekte Funktionalität der Zugleit- und Sicherungssysteme muss durch ein unabhängiges Labor validiert werden[20]. Dabei werden mehrere Testsequenzen, welche aus einzelnen Testschritten zusammengesetzt sind, durchlaufen. Hierzu wird in einem Labor eine reale Zugfahrt simuliert und die Reaktionen der zu untersuchenden Komponente aufgezeichnet. Anschließend werden die Ergebnisse mit den erwarteten Reaktionen verglichen und das Verhalten der Komponente verifiziert. Abschließend wird bei jeder Kampagne, nachdem die Komponenten vollständig verifiziert wurden, ein Bericht erstellt, welcher dem Hersteller und Auftraggeber als Rückmeldung dient. Aufgezeigt werden hierbei alle durchgeführten Tests und das jeweilige Verhalten der Komponenten während den simulierten Fahrten. Treten Fehler bei den Tests auf (siehe Unterkapitel 2.3), werden diese ausführlich in dem Bericht dargelegt.

[1] Deutsch: Europäisches Zugsicherungssystem

Ziel dieser Arbeit ist die Automatisierung der Testberichterstellung zur Vereinfachung des Prozesses und zur Verbesserung der Nutzerfreundlichkeit. Dabei soll das Konzept aus einer vorhergehenden Praxisarbeit [15] angepasst und umgesetzt werden. In diesem Konzept wurde bereits eine Möglichkeit ausgearbeitet, wie Freitexte in den Bericht eingebunden werden können. Zudem soll der Bericht im Hintergrund generiert werden, wenn Änderungen an den Daten vorgenommen wurden. Hierdurch wird dem Nutzer jederzeit ein Bericht zur Verfügung gestellt, ohne dass dieser händisch angefertigt werden muss. Aktuell wird der Bericht auf einem Laborrechner generiert, von welchem aus ein Zugriff zu den Ergebnissen der Testauswertungen besteht. Auf diesem Laborrechner soll der Bericht weiterhin erstellt werden, wobei der Benutzer die Erstellung nicht mehr manuell starten soll. Ein Konzept für die genaue Umsetzung wird in den folgenden Kapiteln entwickelt.

Zur Umsetzung dieser Automatisierung werden zunächst die Grundlagen erläutert, um einen Überblick über das zugrundeliegende System zu erhalten. Im Anschluss wird das bereits bestehende Konzept aufgegriffen und angepasst, da einige Punkte reflektiert und überarbeitet wurden. Darauf aufbauend kann die Automatisierung umgesetzt werden. Hier soll in dem ersten Schritt eine Möglichkeit umgesetzt werden, welche bei der Erstellung des Berichts Abfragen umgeht (siehe Unterkapitel 4.2). Dies bietet eine Grundlage, sodass die Berichterstellung automatisiert werden kann. Zudem wird in diesem Zusammenhang eine einfache Version einer Konfigurationsdatei generiert, um den Zugriff auf die zugrunde liegenden Daten einschränken zu können. Anschließend kann die automatische Überprüfung auf Änderungen in den Daten umgesetzt werden. Ist die Berichterstellung automatisiert, kann die Konfigurationsdatei nochmals genauer betrachtet werden. In dem ersten Schritt liegt der Fokus lediglich auf einer funktionsfähigen Umsetzung, wohingegen in dem nächsten Schritt eine benutzerfreundliche Lösung gesucht und umgesetzt wird. Abschließend gibt es eine Zusammenfassung der Anwendung der Automatisierung und es wird ein Ausblick gegeben, inwiefern das Programm weiterentwickelt werden kann.

2. Grundlagen

Im Folgenden werden zunächst die Grundlagen für die Konformitätstests von Zugsicherungskomponenten erläutert. Dabei wird als erstes auf die Testumgebung eingegangen. Anschließend werden die Tests beschrieben und es wird die Verifizierung der Zugsicherungskomponenten anhand der durchgeführten Tests aufgegriffen. Außerdem wird genauer auf den Aufbau eines Testberichts eingegangen. In diesem Zusammenhang wird der benötigte Inhalt dargelegt. Zudem wird auf den aktuellen Stand der manuellen Berichterstellung eingegangen. Abschließend wird die Automatisierung von Geschäftsprozessen betrachtet. Zuerst geschieht dies im Allgemeinen und anschließend werden Verbindungen zu der betrachteten Berichterstellung hergestellt.

2.1. Die Testumgebung

Zum Testen der Komponenten des ETCS ist ein hierfür akkreditiertes Labor notwendig. Das Institut für Verkehrssystemtechnik am Deutschen Zentrum für Luft- und Raumfahrt in Braunschweig besitzt eines der sechs für diesen Einsatz akkreditierten und allgemein bekannten Labore in Europa. Dies bedeutet, dass das Labor durch eine offizielle Stelle wie z.B. die **D**eutsche **Akk**reditierungs**s**telle (DAkkS) regelmäßig geprüft werden muss. Hierbei wird beurteilt, ob das Labor alle offiziellen Anforderungen erfüllt und weiterhin spezielle Konformitätsbewertungstätigkeiten durchführen darf.[5] Die Gruppe "Technologien" in der Abteilung für "Verifikation und Validierung" führt die Konformitätstests in dem **Rail**way **Si**mulation and **Te**sting (RailSiTe) Labor durch. Dabei handelt es sich um ein eisenbahntechnisches Simulations- und Testlabor. Getestet werden können hier Komponenten in Bezug

auf ihre Konformität und Interoperabilität und das Gesamtsystem in Bezug auf das Zusammenspiel der fahrzeugseitigen und streckenseitigen Komponenten. [20] Für die Zulassung dieser Tests ist eine regelmäßige Akkreditierung notwendig. Hierbei wird überprüft, ob das Labor in der Lage ist, nach der DIN-Norm DIN EN ISO/IEC 17025:2018 Prüfungen durchzuführen. [1]

Das RailSiTe ist in der Abbildung 2.1 zu sehen. Abgebildet sind mehrere Arbeitsplätze, an welchen direkt in dem Labor gearbeitet werden kann. Auf der rechten Seite ist ein Gestell mit einem **D**river **M**achine **I**nterface (DMI) und einem Roboterarm zu sehen. Dieses ist so gebaut, dass hier eine Verdunklung angebracht werden kann. Rechts und links neben dem Gestell sind Serverschränke zu sehen.

Quelle: [19]

Abbildung 2.1.: Das Railway Simulation and Testing Labor

Im Kontext dieser Arbeit wird das Testen von **O**n-**B**oard **U**nits (OBUs) betrachtet. Dabei handelt es sich um Boardcomputer, welche in Zügen installiert sind. Grundlage für die Tests ist die **T**echnical **S**pecification for **I**nteroperability relating to the subsystem "**C**ontrol-**C**ommand and **S**ignalling" (TSI CCS)[17]. Es handelt sich hierbei um die Technische Spezifikation für die Interoperabilität des Teilsystems Zugsteuerung, Zugsicherung und Signalgebung. In dieser Spezifikation wird das ETCS-System beschrieben und es werden Angaben zu den technischen Systemanforderungen, den Schnittstellen zur Technik und weiteren wichtigen Informationen gemacht. [4]

Anhand des TSI CCS können Konformitätstests durchgeführt werden. Hierbei werden

offizielle Testsequenzen erstellt, welche die Richtlinien beinhalten, die die **O**n-**B**oard **U**nit (OBU) erfüllen muss, und welche durch das TSI CCS grundlegend bereits vorgeschrieben werden. Diese können für die verschiedenen Testkampagnen in dem Testlabor verwendet werden. [16]

2.2. Testen der Zugsicherungskomponenten

Bei einer Testkampagne wird zunächst die **O**n-**B**oard **U**nit des Auftraggebers bzw. Herstellers in dem RailSiTe aufgebaut. Dazu werden „eine digitale Signalquelle für die Balisensimulation, ein separater Computer für Videoverarbeitungskomponenten, der Roboter, welcher für die Eingaben am DMI zuständig ist, und eine Kamera, welche die DMI-Aktivitäten aufzeichnet"[16], an den Boardcomputer angeschlossen. Bei dem DMI handelt es sich um das **D**river **M**achine **I**nterface. Dies beschreibt das Display, welches in dem Führerraum installiert ist.[2] In der Abbildung 2.2 ist der Bildschirm aus dem RailSiTe und der Roboterarm für die Eingaben zu sehen. [16]

Quelle: [19]

Abbildung 2.2.: Roboterarm und Bildschirm im RailSiTe

Des Weiteren ist die OBU mit allen benötigten Komponenten in einem Gestell installiert, sodass alles mit einer Verdunklung überzogen werden kann. So kann das DMI gefilmt und eine Bilderkennung angewendet werden, um auf Geschehnisse mit dem Roboterarm reagieren und Aktionen automatisiert auswerten zu können. [16]

Ist die OBU fertig in dem RailSiTe installiert und angeschlossen, können die Testfahrten durchgeführt werden. Hierbei gibt es mehrere Hundert Sequenzen, welche gefahren werden müssen. Die einzelnen Sequenzen bestehen aus mehreren Testschritten. Diese simulieren Situationen, welche auf einer realen Fahrt geschehen können. Dazu zählen unter anderem verschiedene Geschwindigkeitsanzeigen, das Reagieren auf einen Fehler des Lokführers oder die Kommunikation mit streckenseitigen Komponenten.
Durch Automatisierungsskripte können diese Testsequenzen nacheinander gefahren werden. Dabei wird die OBU nach jeder Sequenz neu gestartet und die Sequenzen werden automatisch nacheinander gefahren. Tritt bei einer Fahrt ein Fehler auf, wird die Sequenz automatisch beendet und die nächste Fahrt begonnen. Gespeichert werden bei den einzelnen Sequenzen Log-Einträge, welche bei den Fahrten aufgezeichnet werden. Anhand dieser kann schließlich das Verhalten der OBU verifiziert werden. Zudem wird ein Video des DMI aufgezeichnet, um anschließend manuell prüfen zu können, was während einer Testfahrt passiert ist. [16]

2.3. Verifizierung der Zugsicherungskomponenten

Ein großer Teil des Testverfahrens ist bereits automatisiert. Symbole auf dem DMI können mit Hilfe der Bilderkennung direkt identifiziert und ausgewertet werden. Außerdem existieren verschiedene Softwaretools und Automatisierungsskripte, welche die Log-Einträge mit den gegebenen erwarteten Reaktionen vergleichen und die Schritte anschließend auswerten. Dazu zählt unter anderem das Programm **T**est **S**equence **Eval**uation (TSEval). Dieses wird für jede Sequenz durchgeführt. Dabei werden die Log-Einträge den gegebenen Testschritten zugeordnet. Erstellt werden unter anderem zwei Excel-Tabellen. Eine dieser Tabellen enthält alle Log-Einträge und die entsprechend zugeordneten Testschritte. Bei der zweiten Tabelle handelt es sich um das manuelle Ergebnis, welches händisch angepasst werden kann. Dabei werden die Testschritte den passenden Log-Einträgen zugeordnet, wobei nur die Referenzen zu den jeweiligen Log-Einträgen angegeben werden.[33] Dieser Vorgang ist notwendig, da bei den Durchläufen Fehler auftreten können.

Diese Fehler können in die folgenden drei Gruppen unterteilt werden: [33]

- **Das Testlabor hat einen Fehler:** Dieser Fehler kann entstehen, wenn beispielsweise Komponenten in dem Labor nicht richtig angeschlossen wurden. In diesem Fall muss der Fehler behoben und die Testsequenz erneut gestartet werden.
- **Die ETCS-EVC[1]-Komponente weicht von der Spezifikation ab:** In diesem Fall ist es möglich, dass der Hersteller absichtlich eine andere Reaktion der OBU implementiert hat. Es kann jedoch auch ein nicht erwarteter Fehler entstanden sein, welcher von dem Hersteller korrigiert werden müsste. Daher muss die Ursache des Fehlers untersucht werden, damit dieser ausführlich in dem Testbericht beschrieben werden kann.
- **Die Subset-076-Testsequenz[2] ist fehlerhaft:** Bei der Erstellung der Testsequenz können Fehler entstehen. Eine Möglichkeit ist das falsche Einbinden von Testschritten. Eine andere Möglichkeit ist die Implementierung von ausgewählten Reaktionen. Ist dies der Fall, so ist es möglich, dass die OBU richtig reagiert hat, diese Reaktion jedoch nicht akzeptiert wird, da das Evaluierungsprogramm hiervon keine Kenntnis hat. Dies ist kein konkreter Fehler der Testsequenz, sondern eine abweichende Reaktion der OBU, die den Fehler verursacht.

Wird das Programm TSEval von einem Laborrechner aus ausgeführt, so können mit den richtigen Parametern die Daten direkt in eine Datenbank geschrieben werden. Hierfür müssen der Name der Datenbank, der Benutzername, das Passwort und der SQL-Server in einer Konfigurationsdatei übergeben werden. Anschließend werden die Daten sortiert in eine Datenbank geschrieben. [16]

Am Ende der Verifizierung aller Testsequenzen kann ein Testbericht erstellt werden. Hierbei werden alle Ergebnisse in einem Dokument zusammengefasst. Dazu zählen unter anderem Testschritte, auf welche nicht korrekt reagiert wurde, welche nicht

[1] Der **E**uropean **V**ital **C**omputer (EVC) ist der "sichere Rechnerkern des ETCS-Fahrzeuggerätes"[2].

[2] Bei dem Subset-076 handelt es sich um die Grundlage, auf welcher die Testschritte und -sequenzen erstellt werden [1]. Die einzelnen Dateien des Subsets sind auf der Seite der **E**uropean Union **A**gency for **R**ailways (ERA) [8] zu finden.

implementiert wurden oder welche bei der Zuordnung kommentiert wurden, und die Dokumentation der Fehler. Erstellt wird der Bericht ebenfalls auf einem Laborrechner, da von hier ein Zugriff auf die Datenbank mit den Einträgen möglich ist. Außerdem werden herstellerspezifische Angaben und Informationen zu dem Testlabor aufgezeigt. Der Bericht dient dem Auftraggeber als eine Rückmeldung, wie die zu testende OBU in der Kampagne abgeschnitten hat. Auf die genaue Erstellung des Testberichts wird in Abschnitt 2.5 eingegangen.

2.4. Aufbau eines Testberichts

Der zu erstellende Testbericht hat bei jeder Kampagne den gleichen Aufbau. Dabei handelt es sich bei der ersten Seite des Testberichts um ein Deckblatt, auf welchem folgende grundlegende Informationen zu der Kampagne notiert werden:

- Adresse des Testlabors und des Kunden
- die Richtlinien, nach denen die Prüfung durchgeführt wurde
- Informationen zu dem Bericht: Berichts- und Versionsnummer, Datum
- Name und Unterschrift des Projekt- und Laborleiters

Anschließend folgt ein Inhaltsverzeichnis. Dieses ist in der Abbildung 2.3 zu sehen. Hier wird außerdem deutlich, dass der Bericht in Englisch verfasst wird, um international verständlich zu sein.

Anhand dieses Inhaltsverzeichnisses wird im Folgenden der Inhalt des Berichts geschildert. Hierbei bildet das Verzeichnis eine Leitlinie durch die Inhalte.
Nach dem Inhaltsverzeichnis folgt vor dem ersten Kapitel eine weitere Seite mit allgemeinen Informationen. Dazu zählen die Dokumentenversion, die Autoren des Berichts, eine Änderungshistorie und ein Abkürzungsverzeichnis.

Die ersten beiden Kapitel enthalten einen kurzen Überblick über den Bericht und die Kampagne. Hier ist beschrieben, was und auf welcher Grundlage getestet wurde. Zudem handelt es sich um eine rechtliche Absicherung. Es wird erwähnt, dass dieses

Contents

Quelle: [18]

Abbildung 2.3.: Inhaltsverzeichnis eines Testberichts

Dokument lediglich für den Auftraggeber als Feedback dient und keine Auswertung der Ergebnisse stattgefunden hat. Es wurden ausschließlich die Testfahrten durchgeführt und untersucht, ob die Ergebnisse den Richtlinien entsprechen. In Kapitel 2 ist eine Einleitung zu finden, welche darlegt, um was es in dem Bericht geht. Zudem wird das Testlabor vorgestellt.

2.4.1. Erläuterungen zu der Testausführung

In dem Kapitel *Test Execution* werden Details zu der Durchführung notiert. Dabei wird zuerst beschrieben, was genau getestet wurde. Dazu zählen die genaue Bezeichnung und der Hersteller. Anschließend wird der Umfang betrachtet. Hierzu zählen die Anzahl der durchgeführten Testsequenzen und die Grundlagen, auf denen das

DMI und die **J**uridical **R**ecording **U**nit (JRU) evaluiert werden. Bei der JRU handelt es sich um eine „Einheit zur Aufzeichnung juristisch relevanter Daten"[2], was beispielsweise ein Fahrtenschreiber sein kann. Weiter wird das Testlabor beschrieben, in welchem die Kampagne durchgeführt wird. Das Unterkapitel *Technical Information Regarding the Test Execution* beschreibt, welche Limitierungen von dem System berücksichtigt werden müssen und welche zusätzlichen Aktionen seitens der OBU während der Testdurchläufe aufgefallen sind.

2.4.2. Erläuterungen zu der Testanalyse und den Logdaten

In dem vierten Kapitel geht es um die Testanalyse, nachdem die einzelnen Tests durchgeführt wurden. Hierzu sind zunächst einzelne Definitionen notwendig. Dabei handelt es sich um die einzelnen Kategorien, zu welchen die Testschritte zugeordnet werden. Diese sind genauer beschrieben in der Tabelle 2.1.

Kategorie	**Beschreibung**
passed	Der Schritt wurde ausgeführt und das Ergebnis stimmt mit der gegebenen Spezifikation überein. Ein Schritt, der optional ist und entweder nicht auftaucht oder der Beschreibung entspricht.
passed with comments	Es existieren kleine Abweichungen von der Beschreibung der Testspezifikation.
not passed	Es gibt Abweichungen zu der Spezifikation.
not implemented	Der Schritt existiert nicht in der Testsequenz.
inherited error	Der Schritt kann nicht ausgewertet werden, da das Ergebnis auf dem vorherigen Schritt basiert. Dieser gehört zu einer der Kategorien "not implemented", "passed with comments" oder "not passed".

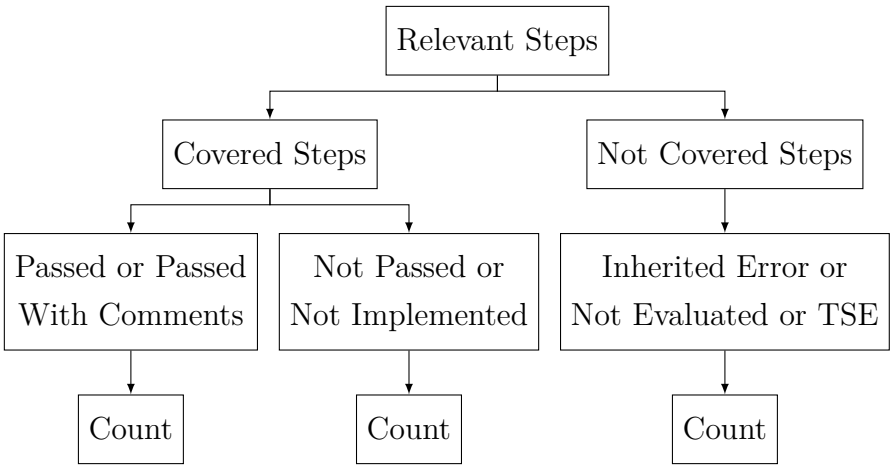
not evaluated	Diese Kategorie zeigt, dass ein Teil der Sequenz nicht getestet oder nicht evaluiert wurde. Dies kommt daher, dass der Schritt von dem Prüfungsrahmen ausgeschlossen ist oder nicht evaluiert werden kann.
test sequence error (TSE)	Es besteht ein Fehler in der Testsequenz.
not relevant	Hierbei handelt es sich um einen informativen Schritt, welcher nicht getestet oder evaluiert werden kann.

Quelle: [18]

Tabelle 2.1.: Beschreibungen der einzelnen Kategorien

Die erste Spalte zeigt die Kategorie, zu welcher die Testschritte bei der Evaluierung zugeordnet werden. In der zweiten Spalte befindet sich zu jeder Kategorie eine Beschreibung. Diese gibt an, in welchen Fällen die jeweiligen Schritte zugeordnet werden.

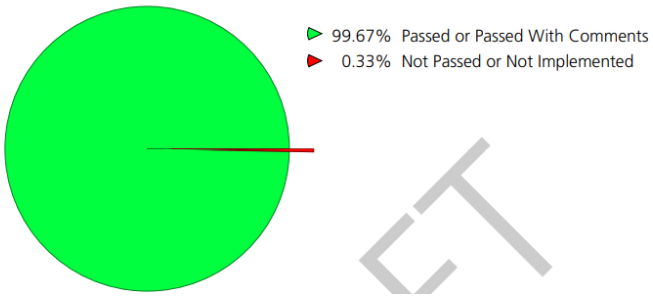
Des Weiteren wird in dem vierten Kapitel eine Zusammenfassung zu der Kampagne erstellt. Dabei handelt es sich um eine Statistik, wie viele relevante Schritte insgesamt evaluiert wurden. Diese Anzahl wird nochmals aufgeteilt in verschiedene Kategorien. Zu sehen ist die Hierarchie in der Graphik 2.4.
Hier ist zu sehen, dass die Schritte zuerst in "Covered Steps" und "Not Covered Steps" aufgeteilt werden. Dabei werden die Schritte, welche nicht ausgeführt wurden, nicht von der Statistik beachtet. Anschließend werden die Schritte nochmals in drei Gruppen aufgeteilt. Die Anzahl der Schritte in den einzelnen Gruppen wird am Ende aufgezeigt. Im weiteren Verlauf wird pro ausgeführter Sequenz eine ähnliche Tabelle erstellt. Außerdem werden die Schritte, welche zu den "Covered Steps" zählen, aber leicht von den Vorgaben abweichen, nochmals aufgelistet.



Quelle: [18]

Abbildung 2.4.: Unterteilung der relevanten Schritte in einzelne Kategorien

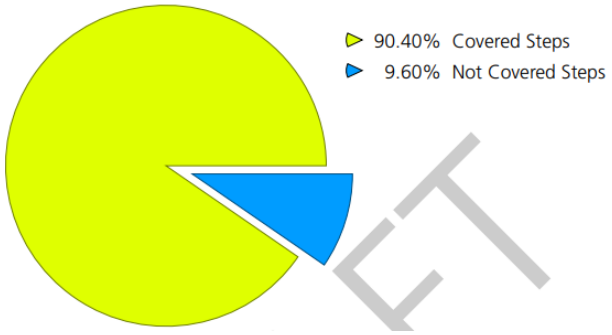
Weitere Statistiken in dem vierten Kapitel sind in den Abbildungen 2.5, 2.6 und 2.7 mit den jeweiligen prozentualen Anteilen zu sehen. In der ersten Statistik werden die "Covered Steps" in die beiden Gruppen aus der Abbildung 2.4 aufgeteilt.



Quelle: [18]

Abbildung 2.5.: Statistik der Conform / Not Conform Steps

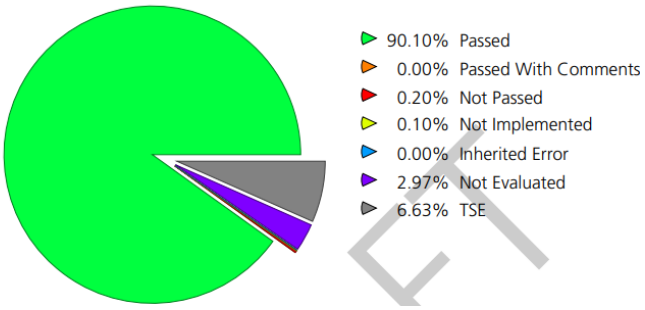
In der zweiten Statistik wird die höhere Ebene betrachtet, die Aufteilung der Schritte in "Covered Steps" und "Not Covered Steps".



Quelle: [18]

Abbildung 2.6.: Statistik der Covered Steps

In der dritten Abbildung werden die Schritte in alle Kategorien aus der Tabelle 2.1 aufgeteilt. Dabei ausgenommen sind die nicht relevanten Schritte.



Quelle: [18]

Abbildung 2.7.: Statistik aller relevanten Schrittkategorien

Wie bereits erwähnt, folgt in dem Kapitel eine Aufteilung der Statistik, welche in Abbildung 2.4 erläutert wurde. Diese Statistik wurde in die durchgeführten Testsequenzen aufgeteilt. Pro Sequenz wird eine eindeutig identifizierbare Nummer angegeben. Außerdem wird die Anzahl der Schritte in den folgenden Kategorien angegeben:

- ”relevant” steps
- ”passed” or ”passed with comments”

- ”not passed” or ”not implemented”
- ”inherited” or ”not evaluated”
- TSE

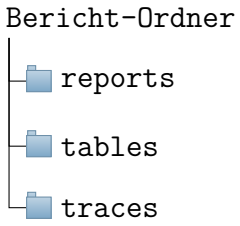
Das letzte Unterkapitel von Kapitel 4 behandelt Schritte, welche von den Vorgaben abweichen. Dazu zählen die Kategorien ”not passed”, ”passed with comments” und ”not implemented”. Pro Sequenz wird dazu eine Tabelle erstellt. Ist in dieser Sequenz ein Schritt zu finden, welcher einer der drei genannten Kategorien zugeordnet werden kann, werden passende Informationen aufgelistet. Dazu zählen die Schrittnummer, die Nummer des Schrittes in dem Testfall, eine Schrittbeschreibung, das Interface von welchem der Schritt ausgeführt wurde und die zugeordnete Kategorie. Zudem wird eine Beschreibung angegeben. Diese ist untergliedert in verschiedene Teile:

- **SITUATION:** Was ist direkt vor dem Event passiert?
- **IS:** Was ist passiert?
- **SHALL:** Was wurde in der Situation erwartet?
- **SEE:** Eine Liste der Anforderungen, welche erfüllt werden müssen.
- **REMARK:** Hierbei handelt es sich um optionale ergänzende Informationen wie z.B. eine Bemerkung, wenn der Test abgebrochen werden musste.

Die Logdaten sind in separaten PDF-Dateien dem Bericht angehängt. Aufgelistet sind die einzelnen Dateien in einer Liste in dem Kapitel 5. Hier ist die Nummer der Testsequenz, der Name der PDF-Datei und eine Prüfsumme für jede Sequenz angegeben.

2.5. Aktueller Stand der Berichterstellung

Für den Bericht existiert eine eigene Ordnerstruktur, sodass alle erstellten und benötigten Daten sortiert abgelegt werden können. Zu sehen ist diese Struktur in Abbildung 2.8.



Quelle: [15]

Abbildung 2.8.: Ordnerstruktur der Testberichterstellung

Im Folgenden wird zunächst die Struktur der Daten beschrieben, welche für die Berichterstellung generiert und benötigt werden. Anschließend wird konkret darauf eingegangen, wie der Bericht erstellt wird.

2.5.1. Struktur der verwendeten Dateien

In dem Ordner *traces* wird pro Sequenz eine Datei abgelegt. Diese enthält eine Tabelle, die in zwei Teile unterteilt ist. Auf der rechten Seite befinden sich alle Log-Einträge. Hier werden verschiedene Details angezeigt:

- Index des Eintrags
- Zeit, zu der der Eintrag aufgezeichnet wurde
- Distanz während der Testfahrt, zu welcher der Schritt aufgenommen wurde
- Modul, von welchem die Aktion durchgeführt wurde
- Nachrichtentyp (Input oder Output)
- übertragene Nachricht
- weitere relevante Variablen

Auf der linken Seite werden die Testschritte abgebildet. Diese werden passend zu den Log-Einträgen zugeordnet. Dabei darf die Reihenfolge der Schritte nicht verletzt werden.
Angegeben werden die folgenden Werte:

- Schrittnummer
- Distanz während der Testfahrt, zu welcher der Schritt aufgenommen wurde
- Schrittnummer in dem Testfall
- Beschreibung des Schrittes
- Modul, von welchem die Aktion durchgeführt wurde
- Richtung, in welche die Nachricht gesendet wird (Input oder Output)
- weitere Details zu dem Schritt
- Ergebnis mit der zugeordneten Kategorie, einem möglichen Kommentar und weiteren Details
- Variablen, welche zur Auswertung benötigt werden

In dem Ordner *tables* werden verschiedene Dateien abgelegt, welche bei der Berichterstellung generiert werden. Dabei handelt es sich um LaTeX-Dateien, welche in den Hauptbericht eingebunden werden. Eine Auflistung aller Dateien ist in der Tabelle 2.2 zu sehen.

Dateiname	**Beschreibung**
bigPicture	In dieser Datei werden Daten festgelegt, welche die Zahlengrundlage für die Statistiken bilden.
checksums	Der Inhalt der Tabelle in Kapitel 5, welche die Namen der Testsequenzen, die Namen der Logdateien und die entsprechenden Prüfsummen enthält.
diagabort	In dieser Datei befinden sich Werte für ein weiteres Diagramm zu der Testausführung. Mit diesem kann der Bericht erweitert werden. Der Standard eines Berichts enthält dieses Diagramm jedoch nicht.
findings	In dieser Datei werden die einzelnen *table...*-Dateien zusammengefügt. Dadurch wird die komplette Tabelle aus Kapitel 4.7 erstellt.

sequences	In dieser Datei sind die einzelnen Sequenzen mit der Versionsnummer und einem Datum aufgelistet. Dabei werden für die Version wie für das Datum Dummy-Werte verwendet. Zudem wird die Datei in dem Bericht nicht verwendet.
steps	Mithilfe dieser Datei wird die Tabelle in Kapitel 4.6 erstellt, welche eine Übersicht über die Testsequenzen und die Kategorien der einzelnen Schritte enthält.
table...	Hierbei handelt es sich um eine Datei, welche pro Sequenz erstellt wird. Darin enthalten ist die Beschreibung der Schritte, welche von den Vorgaben abweichen. Weicht in der Sequenz kein Schritt von den Vorgaben ab, wird zu dieser Sequenz keine Datei erstellt. Die Dateien sind so formatiert, dass sie direkt als Inhalt der Tabelle in Kapitel 4.7 eingebunden werden können.
tracesonly	Diese Datei ist leer, wenn zu jeder Sequenz eine Trace-Datei existiert und keine weiteren Trace-Dateien in dem Ordner liegen. Andernfalls wird hier eine Information notiert. Allerdings wird diese Datei in dem Dokument nicht mehr verwendet.
variablen	In dieser Datei werden beispielsweise die Namen der Datei mit den Kundendaten und der Datei mit den Quellen festgelegt. Außerdem wird hier die Anzahl der betrachteten Sequenzen hinterlegt.

Tabelle 2.2.: Auflistung aller Dateien in dem Ordner *tables*

Der dritte Ordner *reports* enthält die fertigen Berichte in Form von PDF-Dateien. Es können mehrere Berichte existieren, da es sich hierbei um verschiedene Versionen handelt. Zu den Berichten wird außerdem jeweils ein ZIP-Ordner erstellt. Dieser enthält alle Dateien aus dem Ordner *traces*.

In dem Hauptordner liegen das Hauptdokument und weitere LaTeX-Dateien, welche eingebunden werden. Diese Dateien werden in der Tabelle 2.3 aufgelistet und beschrieben. Außerdem liegt hier die Datei "berichtB3.bib", in welcher alle Quellen aufgelistet werden, welche für die Testkampagne benötigt wurden. Diese Datei muss

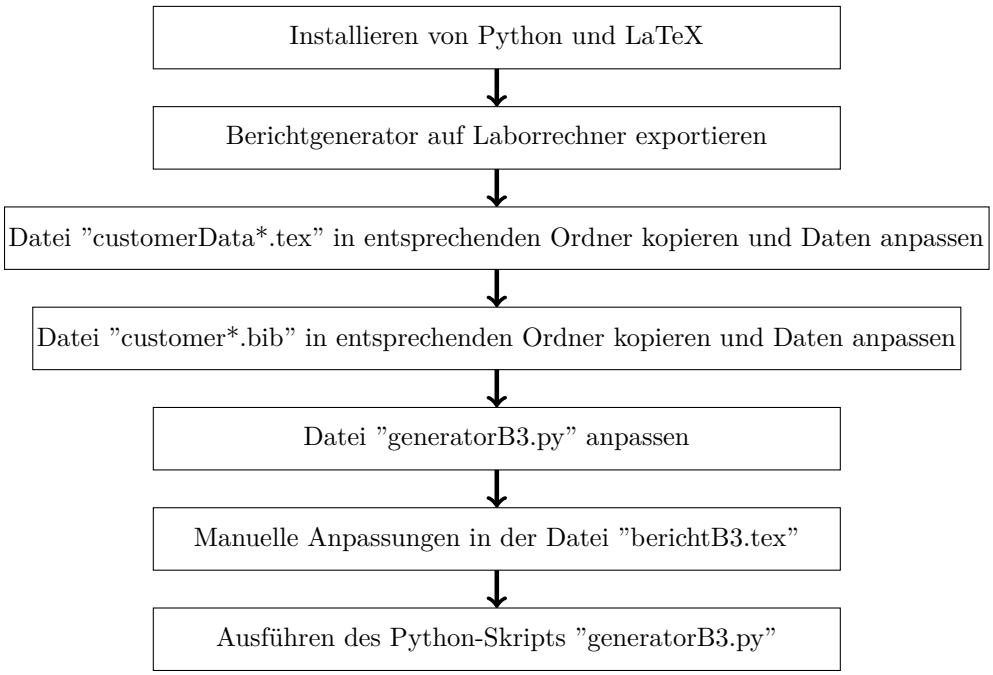
händisch an den Hersteller angepasst werden. Zum Starten der Berichterstellung liegt eine Python-Datei vor. Die genaue Funktion von dieser wird in Unterabschnitt 2.5.2 aufgeführt.

Dateiname	**Beschreibung**
begintraceB3	Mithilfe dieser Datei wird der Anfang der Tabellen formatiert, in welchen die Tracedateien eingetragen werden.
berichtB3	Bei diesem Dokument handelt es sich um das Hauptdokument, in welchem der Bericht zusammengesetzt wird. Hier werden zum Teil die anderen LaTeX-Dateien eingebunden, um beispielsweise Tabellen abzubilden. *(händische Anpassung notwendig)*
customerData...	Der Name dieser Datei wird durch den Namen des Kunden und die Jahreszahl erweitert. Darin enthalten sind alle herstellerspezifischen Angaben, welche in dem Testbericht hinterlegt werden müssen. *(händische Anpassung notwendig)*
endtrace	In dieser Datei werden die Tabellen der Tracedateien beendet.
headfindingsB3	In dieser Datei wird der Beginn der Tabelle in Kapitel 4.7 formatiert. Dazu zählen die Spaltenbreite und die Überschriften der einzelnen Spalten.

Tabelle 2.3.: Auflistung aller Dateien, welche in dem Hauptordner liegen

2.5.2. Ablauf der Berichterstellung

Die Berichterstellung wird auf einem Laborrechner durchgeführt. Von hier aus kann auf die Daten der Evaluierung, welche in einer Datenbank zwischengespeichert sind, zugegriffen werden. Außerdem besteht von hier ein Zugriff auf das interne Projektlaufwerk, welches auch von anderen Laptops aus erreichbar ist. Auf diesem Laufwerk werden die erstellten Berichte gespeichert. In der Abbildung 2.9 ist die Berichterstellung aus der Sicht des Benutzers zu sehen.

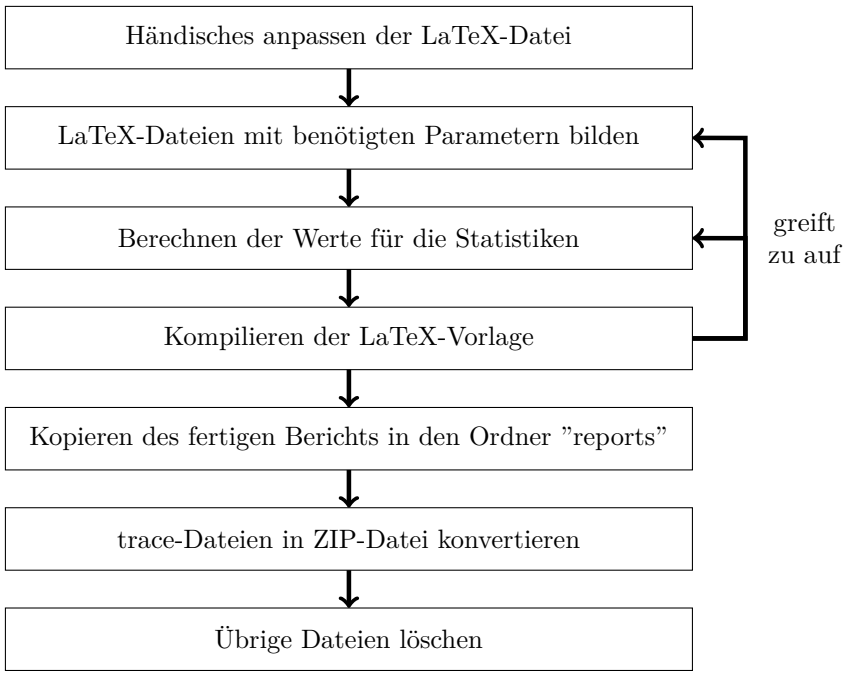


Quelle: [33]

Abbildung 2.9.: Ablauf für das Vorgehen des Nutzers bei der Berichterstellung

Hier ist zu erkennen, dass der Benutzer zuerst die Grunddaten und Skripte kopieren muss. Anschließend können die Kundendaten und die Referenzen an das aktuelle Projekt angepasst werden. In der Generator-Datei müssen die Pfade zu den beiden zuvor angepassten Daten aktualisiert oder hinzugefügt werden und der Datenbankserver und -name wird in der Datei hinterlegt. Anschließend können in der Berichtdatei die Freitexte angepasst werden. Sind alle Daten aktualisiert, wird das Python-Skript ausgeführt. [33] Der Ablauf, welcher dabei durchlaufen wird, ist in der Abbildung 2.10 zu sehen.

Hier werden zunächst mithilfe der Datenbank die verschiedenen Dateien in dem Ordner *tables* erstellt, welche die benötigten Parameter für den Bericht enthalten. Zudem werden die Werte für die Statistiken berechnet und ebenfalls in weitere Dokumente geschrieben. Diese Dokumente und die zuvor händisch angepassten Daten werden in dem Hauptdokument für den Bericht eingebunden. Nach dem Erstellen des Berichts wird dieser zusammen mit einem Zeitstempel in den Ordner



Quelle: [15]

Abbildung 2.10.: Ablaufdiagramm bei der Ausführung des Python-Dokuments

reports kopiert. Die *trace*-Dateien werden zusammen in eine ZIP-Datei konvertiert und ebenfalls in dem Ordner *reports* abgelegt. Alle weiteren erstellten Dateien, welche nicht mehr benötigt werden, können am Ende des Skriptes wieder gelöscht werden.

2.6. Business Process Automation

Automatisierungen spielen in vielen Bereichen von geschäftlichen Prozessen eine immer größere Rolle. Sie werden angewendet, um Risiken zu verringern und Prozesse zu beschleunigen. Durch die Automatisierung können menschliche Fehler verringert werden. Hierzu zählt unter anderem die Belastung der Arbeiter, durch welche Aufgaben vergessen oder nach hinten verschoben werden. Außerdem werden sich wiederholende Aufgaben und Vorgehen durch eine Automatisierung reduziert oder komplett übernommen.[21]

Diese Vorteile lassen sich nach einem Bericht von *Computer Futures* in drei Punkte zusammenfassen: [10]

- **Zeitersparnisse:** Aufgaben werden schneller durch die Übernahmen von Maschinen verarbeitet.
- **Wettbewerbsvorteile:** Mitarbeiter haben mehr Zeit für andere Aufgaben.
- **Kosteneinsparungen:** Personalkosten können gesenkt werden, da durch die Reduzierung der Aufgaben für die Mitarbeiter personelle Ressourcen eingespart werden können. Hierdurch haben die Mitarbeiter mehr Zeit für andere Aufgaben oder die Kosten, welche gespart werden, können in die Automatisierung investiert werden.

Nach *Computer Futures* gibt es zudem sieben Schritte, welche bei einer Prozessautomatisierung betrachtet werden müssen. [10]

1. *Vertrauen in die Technik erzeugen*
   In dem Punkt des Vertrauens besteht das Problem, dass viele Arbeiternehmer Angst vor wegfallenden Arbeitsplätzen oder zu wenig Vertrauen in die Technik im Allgemeinen haben.
   Dies kann jedoch nicht verallgemeinert werden, da in vielen technischen Bereichen eine Automatisierung als Erleichterung der Arbeit angesehen wird und das Vertrauen vor allem in der jungen Generation stark vorhanden ist.
2. *Den Bedarf ermitteln*
   In diesem Punkt muss ermittelt werden, was durch die geplante Automatisierung erreicht werden soll und ob dies wirklich notwendig ist. Hierbei muss auch betrachtet werden, ob es andere Wege gibt, die Aufgaben zu vereinfachen.
   Da die Berichterstellung schon seit längerer Zeit ein Thema in der Abteilung ist, welches viel Zeit in Anspruch nimmt, wurde sich dazu entschieden, den Bericht automatisch erstellen zu lassen. Entsprechend liegt hier eine Optimierung vor, da der Bericht regelmäßig und pünktlich vorliegt und dieser nicht erst in dem letzten Schritt erstellt wird. Genaue Anforderungen, welche die Automatisierung erfüllen soll, werden nochmals in Abschnitt 3.1 dargelegt.

3. *Nach geeigneter Soft- oder Hardware für die Geschäftsprozessautomatisierung suchen*
   In dem Aspekt der passenden Hardware muss geprüft werden, welche Lösungen mit den aktuell bestehenden Verfahren kompatibel sind. Dabei werden frühere Ansätze der Automatisierung betrachtet und in dem Konzept in den Kapiteln 3.2 und 3.3 die zu verwendeten Ressourcen ausgearbeitet.

4. *Eine Strategie zur Umsetzung entwickeln*
   Nachdem ein Konzept erstellt wurde, kann konkret die Umsetzung geplant werden. Dies geschieht in dem Kapitel 5 nochmals detaillierter zusammen mit der anschließenden Umsetzung.

5. *Die **B**usiness **P**rocess **A**utomation (BPA) testen*
   Die Umsetzung muss umfangreich auf alle möglichen Situationen getestet werden. Hierzu wird die entstehende PDF mithilfe eines entwickelten Programmes getestet. Dies wird in Kapitel 4.1 nochmals genauer dargelegt. Zudem werden alle Situationen, welche bei der Automatisierung in der Verwendung auftreten können, händisch getestet.

6. *Die BPA implementieren*
   Werden alle Tests erfolgreich durchgeführt, so kann die Automatisierung in die Prozesse der Abteilung eingebunden werden. Ab der Implementierung in der richtigen Umgebung kann die Automatisierung angewendet werden. Hierbei kann Feedback von den Nutzern gesammelt werden, welches zu einer stetigen Verbesserung der umgesetzten BPA verwendet werden kann.

7. *Planen Sie künftige Neuerungen mit ein*
   Bereits bei der Entwicklung der Automatisierung muss betrachtet werden, dass in der Zukunft neue und bessere Lösungen entwickelt werden. Hierzu kann das Programm modular erstellt werden, sodass es beliebig erweitert werden kann. Zudem wird eine ausführliche Dokumentation angefertigt, sodass der Prozess von beliebigen Mitarbeitern erweitert und ausgebaut werden kann.

In dem Zusammenhang der **B**usiness **P**rocess **A**utomation wird oft von maschinellem Lernen gesprochen. So ist es zum Beispiel auch in dem Artikel von *ComputerWeekly* [24] der Fall. Hier ist von **K**ünstliche **I**ntelligenz (KI)-Werkzeugen die Rede, welche die entwickelten Automatisierungsmodelle trainieren sollen, um eine verbesserte Automatisierung zu erhalten. Jedoch ist nicht in allen Fällen der Automatisierung eine Verbesserung durch **K**ünstliche **I**ntelligenz notwendig oder möglich. In dem in dieser Arbeit betrachteten Anwendungsfall soll regelmäßig ein Bericht erstellt werden, wobei keine Verbesserung des Prozesses durch eine KI möglich ist. Zwischen den unterschiedlichen Kampagnen können sich regelmäßig Strukturen, Texte und Vorgänge ändern. Daher muss die Automatisierung möglichst flexibel gestaltet werden.

Bei den Schritten für die Automatisierung stimmen die Berichte von *Computer Futures* [21] und *ComputerWeekly* zum Teil überein. In beiden Fällen wird zunächst das Bedürfnis bzw. der Bedarf ermittelt, was eine wichtige Grundlage für das weitere Vorgehen bildet. Im weiteren Verlauf sind in dem Artikel von *Computer Futures* die Schritte aufgelistet, welche konkret für die Automatisierung umgesetzt werden müssen. In dem Artikel von *ComputerWeekly* wird auf zwei andere Punkte eingegangen, welche bereits vor der Umsetzung der Automatisierung wichtig sind. Hier wird nochmals aufgegriffen, dass der Entwickler sich zuerst ausführlich mit dem Prozess auseinandersetzen sollte, welcher automatisiert werden soll. Es ist wichtig zu wissen, wie ein Prozess funktioniert, welche Schritte umgesetzt werden müssen und ggf. bereits im Vorfeld bestimmte Prozessschritte zu eliminieren oder zu optimieren, um die Automatisierung zu vereinfachen.

3. Konzept zur Umsetzung der Automatisierung

Im Folgenden wird dargelegt, welche Anforderungen an die Automatisierung gestellt werden. Hierbei sind Nutzererfahrungen involviert, welche beschreiben, was für einen reibungslosen Arbeitsablauf wünschenswert ist. Danach wird auf das bestehende Konzept eingegangen, welches im Rahmen einer Praxisarbeit [15] erstellt wurde. Dieses wird zusammengefasst und nochmals angepasst. Hierbei werden die Laufzeit und der Komfort für den Nutzer bewertet und gegebenenfalls andere Ansätze untersucht.

3.1. Anforderungen an die Automatisierung

Das Ziel der Automatisierung der Testberichterstellung ist eine vereinfachte Verwendung für den Tester. Dabei soll der Bericht regelmäßig automatisch erstellt werden, ohne dass der Tester die Erstellung selbst starten muss.
Zur Einarbeitung der Freitexte und der herstellerspezifischen Daten kann eine Konfigurationsdatei verwendet werden. Diese kann der Tester zu Beginn der Kampagne ausfüllen. Bei der Berichterstellung kann diese dann eingelesen und in den Bericht eingebunden werden. Zudem soll sich der Tester nicht mehr auf den Laborrechner einwählen müssen. Der Bericht soll automatisch im Hintergrund erstellt werden. Dies kann über den Laborrechner realisiert werden. So kann die Berichterstellung im Hintergrund ausgeführt werden, da dies mehrere Stunden dauern kann. Schlussendlich soll die Berichterstellung immer dann gestartet werden, wenn sich Daten auf dem Projektlaufwerk oder in der Datenbank ändern.

Der Vorteil einer regelmäßigen Berichterstellung liegt darin, dass bereits zwischendurch Statistiken eingesehen werden können. Zudem können frühzeitig Fehlermeldungen zu den Sequenzen bearbeitet werden und dem Hersteller können regelmäßig Zwischenberichte geschickt werden. [15]

3.2. Zusammenfassung eines zuvor entwickelten Konzeptes

Im Rahmen eines vorherigen Praxisberichts wurde bereits ein Konzept zur Automatisierung der Testberichterstellung entwickelt [15]. Hierbei wurde sich mit fünf Punkten beschäftigt:

1. Format für den Bericht
2. Format für die Freitexte
3. Programmiersprache
4. Jenkins
5. Die virtuelle Maschine

Für die Auswahl des Formats für den Bericht wurden verschiedene Aspekte betrachtet. Dabei wurden die Formate Microsoft Word Dokument, eine einfache Textdatei und LaTeX miteinander vergleichen. Hierfür wurde die Tabelle 3.1 erstellt, in welcher die drei Anwendungen auf vier Aspekte untersucht wurden.

Anwendung	**Konvertierung in PDF**	**gute Formatierungsmöglichkeiten**	**von extern anpassbar**	**Unabhängigkeit von dem Betriebssystem**
Microsoft Word Dokument	x	x		
Textdatei	x		x	x
LaTeX	x	x	x	x

Quelle: [15]

Tabelle 3.1.: Vergleich verschiedener Formate für den Bericht

In dieser Tabelle ist deutlich zu sehen, dass LaTeX alle Aspekte erfüllt und damit die beste Möglichkeit für den Bericht bildet. Bei LaTeX handelt es sich um eine eigene Programmiersprache, in welcher Dokumente durch Befehle formatiert und einfach in eine PDF-Datei umgewandelt werden können. Zudem wird ausschließlich mit Textelementen gearbeitet. Dies bietet eine gute Möglichkeit, Dokumente durch andere Programme anzupassen. Ein weiterer wichtiger Aspekt ist die Unabhängigkeit von LaTeX. Dokumente können auf verschiedenen Betriebssystemen erstellt werden. Dies ist wichtig, da in dem Institut für Verkehrssystemtechnik die Betriebssysteme Windows und Linux verwendet werden. [15]

Als Format für den Freitext wurde sich für eine Konfigurationsdatei entschieden. Dabei wurde eine einfache Textdatei und eine interaktive PDF betrachtet. Bei der Textdatei müssen viele Fehler abgefangen und die Formatierung sollte bestmöglich eingehalten werden. Die Textdatei ist trotzdem zu Beginn eine gute und einfache Möglichkeit, da die Daten einfach eingelesen werden können. Im Gegensatz hierzu ist die interaktive PDF benutzerfreundlicher, aber das Erstellen und Einlesen ist komplexer. Daher kann eine Textdatei für die ersten Versuche verwendet und später durch eine interaktive PDF ersetzt werden. [15]

Als Programmiersprache wurde Python gewählt, da in dieser Sprache bereits ein Skript zu der Berichterstellung existiert. Zudem ist diese Sprache ebenfalls unabhängig von dem Betriebssystem.
Des Weiteren wird Jenkins verwendet. Dabei handelt es sich um einen Automatisierungsserver, welcher meist in Verbindung mit einem Versionsverwaltungssystem verwendet wird. Sobald eine neue Version eines eingecheckten Projektes in der Versionsverwaltung veröffentlicht wird, wird das Programm kompiliert und eingebundene Tests durchgeführt. [15]

Der letzte Punkt der Automatisierung ist das Umgehen einer manuellen Verbindung zu der virtuellen Maschine, auf welcher der Testbericht erstellt wird. Hierbei handelt es sich um einen Laborrechner, auf welchem sich der Benutzer mit einer Remoteverbindung anmelden muss. Für die Umsetzung kann ein Skript erstellt werden, durch welches eine SSH-Verbindung zu dem Laborrechner hergestellt und dort der Bericht erstellt werden kann. Zur Übertragung der Freitexte kann das Projektlaufwerk

verwendet werden. Die Konfigurationsdatei muss in dem richtigen Ordner abgelegt werden, auf welchen von dem Laborrechner ebenfalls zugegriffen werden kann.
Beachtet werden müssen auch die Fragen, welche bei der Erstellung des Berichts beantwortet werden müssen. Dazu zählen die Abfrage, ob bestehende Trace-Dateien gelöscht und ob der alte Bericht überschrieben werden soll. Die betrachteten Lösungsansätze sind Parameter, welche bei dem Programmaufruf mit übergeben werden können, oder eine weitere Konfigurationsdatei. Eine weitere betrachtete Möglichkeit ist eine Abfrage auf dem Arbeitsrechner, mit welchem die Konfigurationsdatei ausgefüllt werden kann. Dabei besteht allerdings das Problem der automatischen Ausführung bei der Evaluierung. Somit verfällt diese letzte Möglichkeit. [15]

3.3. Anpassung und Erweiterung des Konzeptes

Die ersten drei Aspekte des entwickelten Konzeptes können in der gegebenen Form übernommen werden. Bei der Verwendung von Jenkins kann das Konzept erweitert werden. Neben dem Kompilieren und Testen von eingecheckten Projekten kann Jenkins auch für die Automatisierung verwendet werden. Mithilfe von Jenkins können die Daten auf dem Projektlaufwerk regelmäßig auf Änderungen überprüft werden. Dabei wird in bestimmten Zeitintervallen ein Skript auf der virtuellen Maschine ausgeführt, durch welches überprüft werden kann, ob eine Änderung der Testdaten vorgenommen wurde. Ist dies der Fall, wird die Berichterstellung auf dem Laborrechner gestartet. Bei dem aktuellen Konzept wird der Testbericht von dem Arbeitsrechner aus gestartet. Dies kann zu Performance-Problemen führen, da die Berichterstellung mehrere Stunden dauern kann. Durch die große Anzahl der Testsequenzen in den meisten Kampagnen sind es viele Daten, die verarbeitet werden müssen. Wird die Erstellung von einem anderen Rechner aus gestartet, behindert dies den Nutzer während der weiteren Arbeit nicht. So wird der Bericht immer direkt aktualisiert und der Nutzer kann Statistiken einsehen, wenn die Daten angepasst wurden.

Das Umgehen der virtuellen Maschine wird ebenfalls mit Jenkins umgesetzt. Dadurch ist eine Verbindung von dem Arbeitsrechner aus auf den Laborrechner nicht mehr notwendig, da die Berichterstellung automatisch von dem Laborrechner gestartet wird.

Zur Übertragung der Freitexte und Parameter kann das alte Konzept übernommen werden. Die Schnittstelle ist das Projektlaufwerk, auf welchem Konfigurationsdateien zwischengespeichert werden können. Die Abfragen werden ebenfalls durch die Konfigurationsdatei realisiert. Die Idee der Ausführung der Berichterstellung bei der Evaluierung wurde in der Form verworfen, da hierbei das Problem der Performance auf dem Arbeitsrechner auftritt. Durch eine automatische Ausführung der Berichterstellung im Hintergrund können keine richtigen Abfragen an den Benutzer gestellt werden. Somit wird eine Konfigurationsdatei verwendet.
Jenkins kann jedoch nicht mehr für die Überprüfung der korrekten Berichterstellung in Verbindung mit einem Versionsverwaltungssystem verwendet werden, da von dem Laborrechner aus kein Internetzugriff vorhanden ist. Somit wird eine andere Möglichkeit entwickelt, um die Berichte zu vergleichen. Das Vorgehen hierzu wird in Abschnitt 4.1 beschrieben.

Im Folgenden ist eine Zusammenfassung des fertigen Konzeptes zu sehen. Die fünf Punkte des ursprünglichen Konzepts werden aufgegriffen und die verwendeten Möglichkeiten zusammengefasst.

1. Format für den Bericht: LaTeX
2. Format für die Freitexte: Textdatei / interaktive PDF
3. Programmiersprache: Python
4. Automatische Testberichterstellung: Jenkins
5. Die virtuelle Maschine: Vermeiden durch Jenkins

4. Umsetzung der Konfigurationsdatei in Form eines Textdokuments

In dem ersten Schritt der Umsetzung soll die Berichterstellung vereinfacht werden, indem manuelle Eingaben auf ein Dokument beschränkt werden. Dabei entsteht der folgende Ablauf 4.1 für den Nutzer:

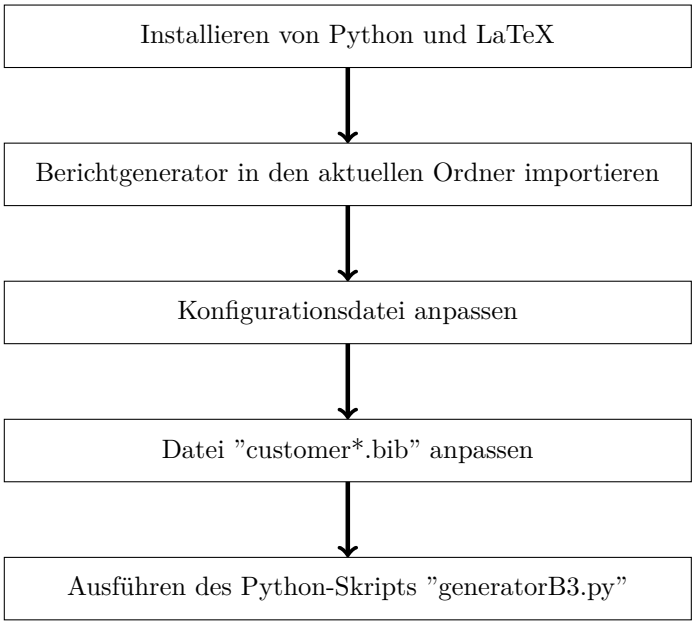


Abbildung 4.1.: Ablaufdiagramm nach der Anpassung des Projektes

Hierbei fallen zwei Schritte weg und der Benutzer muss nur noch eine Datei für die Freitexte und Parameter anpassen. So müssen die einzelnen Parameter und Angaben nicht mehr in verschiedenen Dateien gesucht werden. Dabei wird zuerst eine Textdatei für die Übergabe der Daten verwendet, welche während der Berichterstellung benötigt und pro Kampagne individuell angepasst wird. Das Format für die Daten kann später angepasst werden, indem eine interaktive PDF generiert wird. Für den ersten Schritt der Automatisierung ist jedoch eine Textdatei ausreichend.

Im Folgenden wird zuerst eine Methode entwickelt, wie das Ergebnis während der Entwicklung des Berichtgenerators regelmäßig getestet werden kann. Anschließend wird die Umsetzung der Textdatei für verschiedene Parameter geschildert.

4.1. Testen der Anpassungen

Wird der Code für den Bericht angepasst, muss überprüft werden, ob das Ergebnis weiterhin korrekt ist. Hierzu wurde ein externes Programm geschrieben, welches zwei PDF-Dateien vergleichen kann. Dabei werden die beiden Dateien zunächst eingelesen und in eine Textdatei geschrieben. Dies geschieht getrennt für beide Dateien. Die Funktion, welche in dem Code 4.1 zu sehen ist, wurde für das Übertragen des Inhalts der PDF in eine Textdatei erstellt. Die Übergabeparameter sind hierbei der Name des PDF-Dokuments und der Name der zu erstellenden Textdatei.

```
def writeToTxt(pdfName, txtName):
    # open the pdf
    pdf = open(pdfName, "rb")
    pdf_fileReader = PyPDF2.PdfFileReader(pdf)

    # write first pdf to txt
    with open(txtName, 'w', encoding="utf-8") as f:
        for page in pdf_fileReader.pages:
        f.write(page.extractText())

    f.close()
    pdf.close()
```

Listing 4.1: Übertragen des Inhalts in eine Textdatei

Als erstes werden die PDF- und die Textdatei geöffnet. Anschließend kann der Inhalt seitenweise in das Textdokument geschrieben werden. In dem letzten Schritt werden beide Dokumente wieder geschlossen, um weitere Zugriffe zu ermöglichen. [34]

Anschließend können beide Textdateien verglichen werden. Dazu werden beide geöffnet und zeilenweise verglichen. Sind die Zeilen nicht gleich, so werden die Zeilennummer und die jeweilige Zeile der beiden Dokumente ausgegeben. [11] Wurde das gesamte Dokument verglichen, wird berechnet, wie groß die Differenz der Anzahl der Zeilen ist. Ist die Differenz von den Zeilen der ersten Datei und den Zeilen der zweiten Datei kleiner als 1, werden die Zeilen nochmal in anderer Reihenfolge verglichen. Dabei werden hier zuerst die Zeilen des zweiten Dokumentes betrachtet und mit den Zeilen des ersten Dokumentes verglichen. Anschließend wird die Differenz der Zeilen des zweiten Dokuments und der Zeilen des ersten Dokuments berechnet. Die Differenz wird am Ende des Programmes ausgegeben, um übrige Zeilen aufzuzeigen, welche nochmals händisch überprüft werden können. Am Ende werden beide Textdateien wieder geschlossen und das Programm beendet.

Nach jeder größeren Änderung des Programmes wird die neue PDF-Datei mit einer älteren verglichen. So kann sichergestellt werden, dass bei den Anpassungen keine Fehler eingebaut werden. Werden alle Dateien neu erstellt, so werden neue Prüfsummen für die einzelnen Logdateien generiert. Daher werden diese Zeilen bei dem Vergleich der beiden PDF-Dateien angezeigt. Dies ist jedoch zu vernachlässigen, da dies nicht auf falsche Anpassungen des Programmes zurückzuführen ist.

4.2. Aktueller Stand in Bezug auf die Parameter und Freitexte

Der Benutzer muss vor der Berichterstellung die Dateien "customerData*.tex", "generatorB3.py" und "berichtB3.tex" anpassen. In der ersten Datei "customerData*.tex" werden die Parameter, welche in dem Bericht verwendet werden, als Kommandos eingebunden. Hierbei handelt es sich um Parameter wie Kundendaten, Versions-

nummern, Namen und weitere Informationen zu dem Bericht. Aufgebaut sind die Kommandos wie in dem Codeabschnitt 4.2.

```
\newcommand{\parameter}{value}
\newcommand{\Date}{22 July 2021}
```

Listing 4.2: Beispielhafte Kommandozeilen

In der ersten Zeile ist der allgemeine Aufbau eines Kommandos zu sehen. Dieses beginnt mit einem Backslash, auf welchem ein Parameter folgt. In der hinteren Klammer ist der Wert zu sehen, welcher anstelle des Parameters in den Bericht eingebunden werden soll. In der zweiten Zeile ist ein konkretes Beispiel zu sehen. Das angegebene Datum kann mit dem Kommando "\Date" in den Text eingebunden werden. [23]
In der Datei "generatorB3.py" werden die Daten für die Verbindung zu der Datenbank angegeben. Dabei handelt es sich um den Namen der Datenbank, den Host, einen Port, den Usernamen und das Passwort. Zudem müssen die Namen der Datei mit den Parametern und der Bibliotheksdatei angepasst werden.
Die dritte Datei "berichtB3.tex" beinhaltet den Hauptinhalt des Berichts. Hier können die Freitexte und kleine Parameter wie das Datum angepasst werden.

Zu dem aktuellen Stand zählen außerdem drei Fragen, welche bei der Ausführung des Python-Skriptes beantwortet werden müssen.

1. x old pdf-trace(s) found. Re-use them? (ATTENTION: old pdf-traces may not match the data in the database!) [yes]/no:
2. Create new pdf-traces? [no]/yes:
3. Compile final pdf-file? [yes]/no:

Für die drei Abfragen gibt es eine gemeinsame Methode, welche für eine Ausgabe auf dem Terminal wie auch für eine Benutzereingabe verwendet wird. Abgebildet ist die Methode in dem Codeausschnitt 4.3.

```
def confirm(prompt=None, default=False):
    if prompt is None:
        prompt = 'Confirm'
    if default:
        prompt = '%s [%s]/%s: ' % (prompt, 'yes', 'no')
    else:
        prompt = '%s [%s]/%s: ' % (prompt, 'no', 'yes')

    while True:
        ans = raw_input(prompt).lower()
        if not ans:
            return default
        if ans == 'yes' or ans == 'y':
            return True
        if ans == 'no' or ans == 'n':
            return False
        print 'Please answer yes or no.'
```

Listing 4.3: Alte Methode für die Konsolenausgabe und Benutzereingabe

Als Übergabeparameter werden ein Text für die Ausgabe auf dem Terminal und ein Standardwert angegeben. Diese beiden Parameter werden formatiert auf der Konsole ausgegeben. Ist der Standardwert ein *True*, steht in der Ausgabe das "yes" vorne und an zweiter Stelle ein "no". Bei der Übergabe eines *False* wird die Reihenfolge der Ausgabe gewechselt. Anschließend wird auf eine Eingabe von dem Benutzer gewartet. Wird die Frage positiv beantwortet, ist der Rückgabewert ein *True*. Bei einer negativen Antwort wird ein *False* zurückgegeben. Die Benutzereingabe ist möglich, bis der Benutzer als Eingabe einen der folgenden Werte gibt: "yes", "y", "no", "n".

4.3. Erstellen und Einlesen einer Konfigurationsdatei

Als Konfigurationsdatei wird vorläufig eine Textdatei gewählt. Diese ist unterteilt in einzelne Abschnitte, getrennt durch den Separator "#####". Die Namen der einzelnen Abschnitte befinden sich direkt hinter dem Separator, getrennt mit einem Doppelpunkt zu dem Inhalt des Abschnitts.
In dem ersten Abschnitt werden die Parameter für die Erstellung des Berichts

angegeben. Dazu zählen die Daten für die Verbindung zu der Datenbank und der Parameter *CustomerFiles*. Mithilfe dieses Parameters werden die Datei mit den Kommandos zu den einzelnen Parametern und die Datei mit den Quellen benannt. Für die Automatisierung müssen außerdem die Fragen bei der Berichterstellung durch die Konfigurationsdatei beantwortet werden. Hierzu werden drei Parameter in dem ersten Abschnitt hinzugefügt. Diese sind in dem Codeausschnitt 4.4 zu sehen.

```
reusePDFtraces: yes
newPDFtraces: no
compilePDFfile: yes
```

Listing 4.4: Konfigurationsdatei für die Abfragen

Der zweite Abschnitt enthält Parameter für den Text in dem Bericht. Hier werden unter anderem die Daten zu dem Kunden, Versionsnummern und verschiedene Namen der Beteiligten benannt. Diese Daten wurden bisher in der Datei "customerData*.tex" gespeichert. Innerhalb der ersten beiden Abschnitte werden die Parameter zeilenweise angegeben. Auch hierbei wird der Parameter mit einem Doppelpunkt von dem Wert getrennt.
Anschließend folgt ein Abschnitt mit der Änderungshistorie. Hier werden die einzelnen Versionen aufgelistet. Dabei werden die Versionsnummer, das Datum und eine Beschreibung angegeben.
Ab dem vierten Abschnitt werden die Inhalte der Freitexte angegeben. Dabei werden für alle Kapitel, in welchem sich Freitexte befinden können, einzelne Abschnitte festgelegt. Innerhalb der Abschnitte befindet sich jeweils ein Textblock, welcher so in den Bericht eingebunden wird. Hierbei sind die Zeilenumbrüche genau so in dem Bericht zu übernehmen. Die Freitexte beziehen sich auf Kampagnen, in welchen nur ausgewählte Testsequenzen durchgeführt werden. In diesen Fällen müssen bestimmte Textpassagen angefügt werden, welche zeigen, dass die Sequenzen von dem Hersteller ausgewählt wurden und dass der Test keinen vollständigen Konformitätstest repräsentiert. Hierbei dürfen die Abweichungen, die der Kunde verlangt, "keine Auswirkung auf die Integrität des Laboratoriums oder die Validität der Ergebnisse haben"[33]. Handelt es sich bei der Kampagne nicht um eine Delta-Kampagne[1],

[1]Eine Kampagne, bei welcher nur ausgewählte Testsequenzen behandelt werden.

so sollen diese Texte nicht in dem Bericht erscheinen. Hierzu wird in dem ersten Abschnitt ein weiterer Parameter zu den drei Fragen hinzugefügt. Ist hier ein "yes" angegeben, so werden die Texte in den Bericht eingebunden. Bei einem "no" werden die Texte ausgelassen.

Eingelesen werden die Daten aus der Textdatei als ein einzelner String. Dieser kann anschließend an dem Separator getrennt werden. Gespeichert werden die Daten in verschiedenen Dictionaries. Hiervon werden drei angelegt. Ein Dictionary mit den Abschnitten der Textdatei, eins mit den Parametern für die Erstellung des Berichts und eins mit den Parametern für den Bericht direkt. Für das Speichern der Daten in einem Dictionary wird die Funktion 4.5 verwendet.

```
def writeToDictionary(data, dictionary, forReport):
    count = 0
    for line in data:
        data[count] = line.split(': ', 1)
        if len(data[count]) == 2:
            parameter = data[count][0]
            entry = data[count][1]
            if not entry:
                print('Parameter '+parameter+' not found.')
            else:
                if forReport:
                    entry = entry.replace('_', r'\_')
                dictionary[parameter] = entry
            count += 1
```

Listing 4.5: Funktion zum Übertragen der Daten in ein Dictionary

Die Übergabeparameter sind die Daten, welche bereits in die einzelnen Abschnitte bzw. Parameter, das Dictionary, in welches die Daten übertragen werden sollen, und ein Boolean-Wert unterteilt wurden. Dieser gibt an, ob die Daten in den Bericht eingebunden werden sollen, um weitere Anpassungen an den Textelementen vornehmen zu können. In der Funktion werden die einzelnen Einträge in der übergebenen Liste mit den Daten durchgegangen. Hierbei werden die Einträge an dem ersten Doppelpunkt in einen Parameter und den entsprechenden Eintrag aufgeteilt. Existiert ein Wert, kann dieser in das Dictionary übertragen werden. Zuvor wird der Wert angepasst, wenn als Boolean ein *True* übergeben wird. Hierbei werden beispielsweise

Unterstriche maskiert, indem ein Backslash vor das Zeichen gesetzt wird. [26]
Diese Funktion wird für das Abspeichern der Abschnitte, der Parameter für die Berichterstellung und der Parameter für den Bericht selbst verwendet.

Auszüge aus der erstellten Textdatei sind in dem Abschnitt A.1 abgebildet. Die erste Abbildung A.1 enthält die ersten beiden Abschnitte mit den Parametern zu der Berichterstellung und für das Einbinden in den Bericht. Die Abbildung A.2 beinhaltet die Modifikationshistorie und die einzelnen Freitexte.

4.4. Einbinden der Daten

Um die Parameter für die Berichterstellung in das Skript einzubinden, wird auf die Einträge in dem Dictionary zugegriffen. So können die konkreten Werte beispielsweise für die Verbindung zu der Datenbank einfach ersetzt werden. Weiter gibt es den Parameter "customerFiles". Dieser setzt sich aus dem Firmennamen des Auftraggebers und der Jahreszahl zusammen. Verwendet wird der Parameter für die Konfigurationsdatei in Form eines LaTeX-Dokuments und für die Bibliotheksdatei. Der Aufbau der beiden Namen ist in Codeausschnitt 4.6 zu sehen.

```
ConfigDataFileName = 'tables/customerData' +
    parameterCreationDict['CustomerFiles'] + '.tex'

ConfigBibFileName = 'customer' + parameterCreationDict['
    CustomerFiles'] + '.bib'
```

Listing 4.6: Erstellen der Dateinamen

Anschließend werden die Namen der beiden Dateien in der Datei *variable.tex* als Kommandos hinterlegt. Das Schreiben des Kommandos in die Datei erfolgt mit der Zeile in dem Codeabschnitt 4.7.

```
file.write("\\newcommand{\\"+parameter+"}{"+value+"}\n")
```

Listing 4.7: Erstellen einer Kommandozeile in der LaTeX-Datei

Für die Beantwortung der Fragen bei der Berichterstellung muss die Funktion in dem Codeausschnitt 4.3 angepasst werden. Die angepasste Version ist in dem Codeausschnitt 4.8 abgebildet.

```
def confirm(answer=None, default=False):
    if answer is None:
        print('No answer given.')
        return default
    if answer == 'yes' or answer == 'y':
        print('Answer: Yes')
        return True
    if answer == 'no' or answer == 'n':
        print('Answer: No')
        return False
    print('No correct answer given. Please select yes or no
       the next time.')
    return default
```

Listing 4.8: Angepasste Methode für die Verarbeitung der Antworten aus der Konfigurationsdatei

Als Übergabeparameter werden hier die eingelesene Antwort und eine Standardantwort übergeben. In der Funktion wird der zweite Teil der alten Funktion beibehalten. Wird keine Antwort in der Textdatei angegeben oder entspricht die Antwort nicht einer der vier erwarteten Antworten, wird die Standardantwort übernommen. Andernfalls wird überprüft, ob die Antwort einem der vier Strings entspricht: "yes", "y", "no", "n". Entsprechend der angegebenen Antwort wird ein Boolean-Wert zurückgegeben.

Die Parameter, durch welche Text in den Bericht eingebunden werden soll, werden als einzelne Kommandos in die Datei *customerData*.tex* geschrieben. Dies geschieht ebenfalls mit der Zeile in dem Codeausschnitt 4.7. Zudem werden zwei weitere Befehle generiert. Zum einen wird die Adresse, welche in vier einzelnen Parametern angegeben wird, zu einem Parameter zusammengefasst. Zum anderen wird ein Befehl für den Dokumentennamen erstellt. Dieser setzt sich aus dem Anfang "Bericht-" und der Berichts-ID zusammen. Diese wurde zusammen mit den anderen Parametern in der Textdatei abgefragt. Ein weiteres Kommando beinhaltet die Änderungshistorie. Der eingelesene Text wird zunächst bearbeitet. Hierbei wird die Leerzeile zu Beginn des Eintrags entfernt. Außerdem wird die Historie durch ein Semikolon beendet.

Daher können die Leerzeilen am Ende entfernt werden, indem ab dem Semikolon alle Zeichen entfernt werden. In dem Text selbst werden die Doppelpunkte zwischen der Versionsnummer und der Beschreibung durch ein Und-Zeichen ersetzt und die Zeilenumbrüche in doppelte Backslashes gewechselt. Hierdurch wird der Text so formatiert, dass er als Inhalt einer Tabelle in das LaTeX-Dokument eingebunden werden kann. In dem Codeausschnitt sind beide Zeilen zu sehen.

```
v1.0, 11/02/2022: Initial Version

v1.0, 11/02/2022 & Initial Version\\
```

Listing 4.9: Einbinden der Änderungshistorie

Die erste Zeile zeigt, wie eine Version in der Textdatei angegeben wird. In der zweiten Zeile ist der formatierte Eintrag zu sehen, welcher in dem Kommando hinterlegt wird. Dieser entspricht der Form eines Tabelleneintrags in LaTeX.
Die Datei *customerData*.tex* wird in der angepassten Version der Berichterstellung in dem Ordner *tables* gespeichert. Für die Verschiebung wurde sich entschieden, da sich in diesem Ordner alle Dateien befinden, welche während der Berichterstellung generiert werden. In diesen Dateien müssen keine Anpassungen vorgenommen werden.

Aus den weiteren Abschnitten der Textdatei werden einzelne LaTeX-Dateien erstellt. Diese werden in einem weiteren Ordner namens *content* abgespeichert. Um die Texte in die entsprechenden Dateien zu schreiben, wird auf das erste Dictionary mit den Abschnitten zugegriffen. Das Vorgehen ist in dem Codeabschnitt 4.10 zu sehen.

```
lenSections = len(sections)
for index in range(4, lenSections):
    item = sections[index]
    fchap = open('content/' + item + '.tex', 'w')
    if deltaCampagne:
        fchap.write(data[item])
    fchap.close()
```

Listing 4.10: Erstellen der LaTeX-Dateien für die einzelnen Kapitel

In diesem Code wird zunächst die Größe der Liste bestimmt, in welcher die Überschriften der einzelnen Abschnitte abgespeichert sind. Durch den Separator zu Beginn

des Dokuments ist der erste Abschnitt in dem Text leer. Anschließend folgen die zwei Abschnitte mit den Parametern und der Abschnitt mit der Modifikationshistorie. Erst ab dem fünften Abschnitt beginnen die Inhalte der einzelnen Kapitel. Daher wird in der for-Schleife erst ab dem Index vier begonnen. Innerhalb der Schleife wird der Titel des Abschnittes ermittelt und anschließend der Inhalt in eine entsprechende LaTeX-Datei geschrieben, wenn in der Konfigurationsdatei angegeben wurde, dass es sich um eine Delta-Kampagne handelt. Am Ende wird die Datei wieder geschlossen, um bei der Berichterstellung auf den Inhalt zugreifen zu können.
In dem Hauptbericht können anschließend die einzelnen Dateien eingebunden werden. Dies geschieht beispielsweise mit dem Befehl `\input{content/chapter1}`. Dadurch wird der Text, welcher in der angegebenen Datei hinterlegt ist, in den Bericht eingebunden.

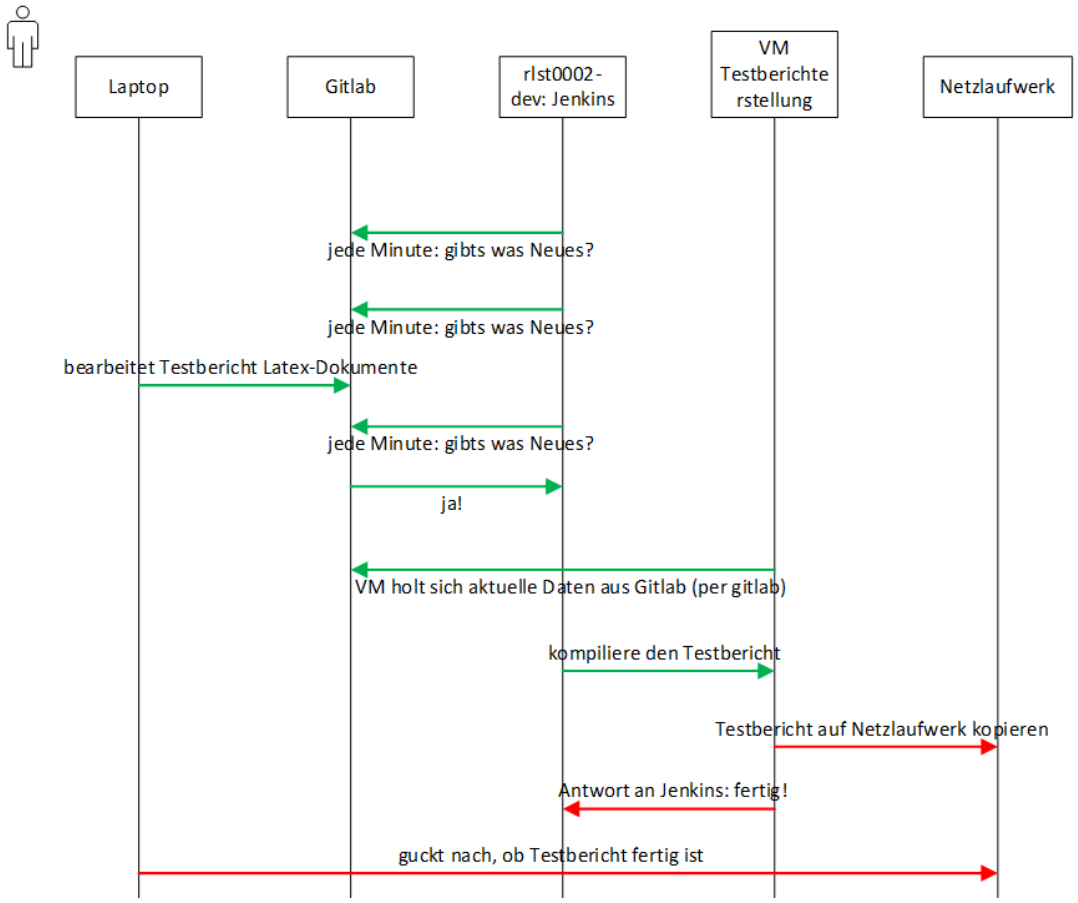
5. Automatisierung der Berichterstellung

Für die automatische Erstellung des Testberichts wird Jenkins verwendet. Anhand von Jenkins können die Daten einer Kampagne regelmäßig kontrolliert werden. Dabei können beispielsweise Versionsverwaltungssysteme verwendet werden, sodass bei jeder veröffentlichten Änderung der Bericht neu erstellt wird. Eine weitere Möglichkeit ist, in regelmäßigen Zeitintervallen einen Befehl auszuführen.
Im Folgenden wird zunächst der aktuelle Stand beschrieben, da bereits ein Ansatz zu Teilen geplant und umgesetzt wurde. Anschließend wird das aktuelle Konzept entwickelt und beschrieben. Anhand dieses Konzeptes kann die Automatisierung umgesetzt werden.

5.1. Aktueller Stand der Automatisierung

Die Automatisierung der Berichterstellung wurde bereits in Ansätzen umgesetzt. Dabei wurden drei Komponenten verwendet: ein GitLab-Repository, ein Jenkins-Projekt und ein Arbeitsverzeichnis auf einer virtuellen Maschine. Die Arbeitsweise der Berichterstellung ist in der Abbildung 5.1 zu sehen.

Wird ein Dokument auf dem Laptop bearbeitet, gelöscht oder hinzugefügt und diese Änderung bei GitLab hochgeladen, wird dies von Jenkins bemerkt. Hierbei wird mit Polling gearbeitet. Dabei handelt es sich um eine "zyklische Abfrage"[9], durch welche jede Minute überprüft wird, ob Änderungen an dem Repository vorliegen. Werden Änderungen gefunden, führt Jenkins die in der Konfiguration angegebenen



Quelle: [22]

Abbildung 5.1.: Ablaufdiagramm des aktuellen Stands der Berichterstellung

Shell-Befehle aus. Dabei werden die aktuellen Daten aus GitLab auf die virtuelle Maschine geladen und der Testbericht wird kompiliert. Bis zu diesem Punkt wurde die Automatisierung bereits umgesetzt. Noch nicht realisiert sind die letzten drei Schritte. Der fertige Testbericht muss nach der Erstellung auf das Netzlaufwerk kopiert werden und von der virtuellen Maschine kann an Jenkins die Nachricht gesendet werden, dass die Berichterstellung beendet wurde. Anschließend kann von dem Laptop aus überprüft werden, ob der Testbericht korrekt erstellt wurde. [22]

5.2. Konzept

Zur neuen Umsetzung der Automatisierung wird zunächst ein Konzept für die Umsetzung entwickelt. Das Vorgehen bei der Umsetzung ist in der Abbildung 5.2 zu sehen.

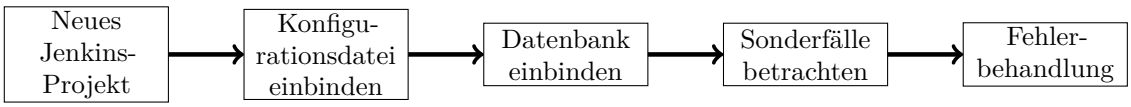
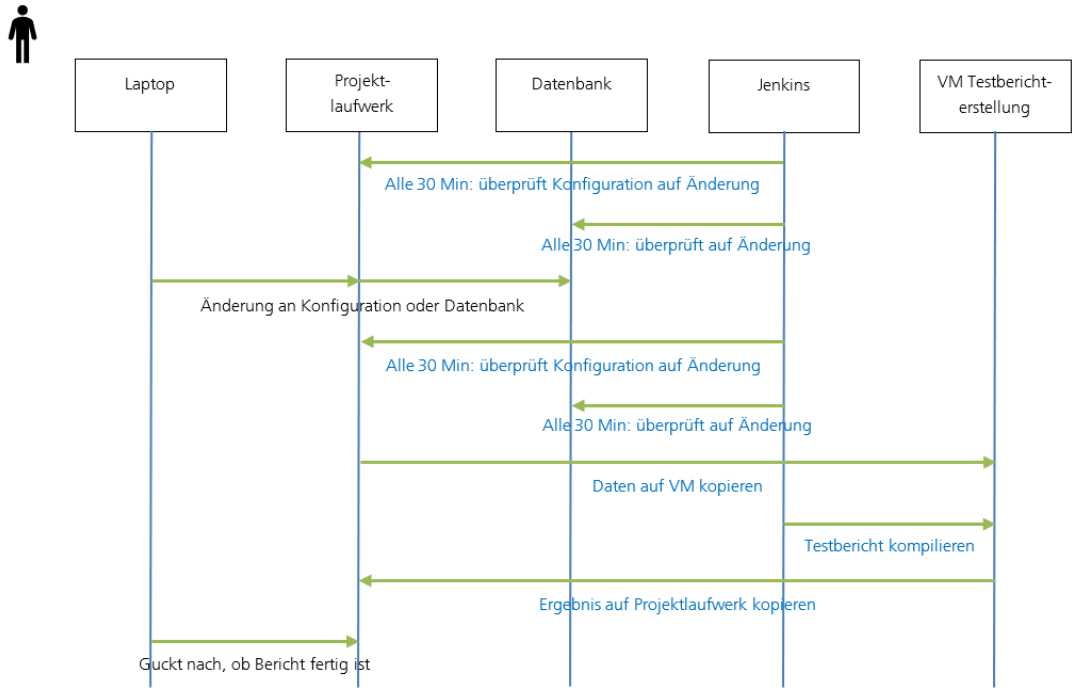


Abbildung 5.2.: Ablaufdiagramm zu der Umsetzung der Automatisierung

In dem ersten Schritt soll ein neues Projekt in Jenkins erstellt werden, anhand von dem die Automatisierung neu umgesetzt werden kann. Zur Automatisierung soll hier jedoch kein Versionsverwaltungssystem eingebunden, sondern in bestimmten Zeitintervallen ein Skript ausgeführt werden. Dieses Skript soll auf der virtuellen Maschine erstellt werden, auf welcher sich der Berichtgenerator befindet. Von Jenkins aus soll diese virtuelle Maschine als Arbeitsort verwendet werden. Bei der Erstellung des Skripts soll zunächst die Konfigurationsdatei eingebunden werden. Diese soll auf das Änderungsdatum überprüft werden. Wurde die Datei verändert, nachdem der Bericht das letzte Mal erstellt wurde, kann der Bericht neu gebildet werden. Anschließend kann die Datenbank in das Skript eingebunden werden. Auch hier soll überprüft werden, ob Änderungen an den Daten vorliegen seitdem der Bericht das letzte Mal erstellt wurde. Zeitgleich mit dem Skript wird eine weitere allgemeine Konfigurationsdatei erstellt, in welcher die Kampagnen aufgelistet werden, von denen der Bericht automatisch erstellt werden soll. Dabei muss für eine Kampagne der Pfad zu dem Ordner des Berichtgenerators angegeben werden. Die Pfade der Kampagnen können erweitert werden durch das Datum der letzten Berichterstellung. Anhand dieses Datums können die Konfigurationsdatei und die Datenbank auf Änderungen überprüft werden.
Das neue Ablaufdiagramm hierzu ist in der Abbildung 5.3 zu sehen. Zu sehen sind in blauer Schrift alle Schritte, welche durch das Skript umgesetzt werden. Der Benutzer muss lediglich Daten ändern, sodass ein neuer Bericht erstellt wird.



Quelle: [14]

Abbildung 5.3.: Ablaufdiagramm des neuen Stands der Berichterstellung

Nach der Umsetzung der allgemeinen Automatisierung müssen folgende Sonderfälle betrachtet werden:

1. *Berichterstellung wird nicht erfolgreich umgesetzt:*
   Für den Fall, dass bei einer Berichterstellung Fehler auftreten und der Bericht nicht erfolgreich erstellt werden kann, soll der Bericht in einen lokalen Ordner kopiert werden. Hierzu existiert ein Jenkins-Ordner auf der virtuellen Maschine, welcher dazu verwendet werden kann. Nach der Erstellung können die benötigten Dateien zurück auf das Projektlaufwerk kopiert werden. Außerdem müssen die Fehler bei der Anwendung von LaTeX abgefangen werden, da hierdurch das Programm nicht bis zum Ende ausgeführt wird und eine Interaktion erforderlich ist.

2. *An den Daten mehrerer Kampagnen wurden Änderungen vorgenommen:*
   Müssen für mehrere Kampagnen neue Testberichte erstellt werden, kann Threa-

ding verwendet werden. Dabei werden die Testberichte zeitgleich erstellt und die vollen Ressourcen der virtuellen Maschine werden verwendet.

3. *Änderungen an einer Kampagne in kurzen Zeitabständen:*
   Werden innerhalb von kurzen Zeitabständen Änderungen an den Daten vorgenommen, so wird der Bericht zeitgleich mehrfach erstellt, wenn die Erstellung länger als eine halbe Stunde andauert. Um dies zu umgehen, kann eine temporäre Datei in dem Ordner der entsprechenden Kampagne erstellt werden, wenn die Berichterstellung begonnen wird. Am Ende wird diese wieder gelöscht. Zu Beginn der Berichterstellung kann so überprüft werden, ob die temporäre Datei in dem Ordner vorhanden ist. Ist dies der Fall, so wird die Berichterstellung nicht gestartet und die Zeiten werden nicht verändert. Dadurch kann der Bericht bei der nächsten Durchführung neu erstellt werden, wenn zu diesem Zeitpunkt die temporäre Datei nicht mehr vorhanden ist.
   Tritt ein Fehler auf, so muss mit hoher Wahrscheinlichkeit etwas an der Konfigurationsdatei angepasst werden, da der Benutzer keine anderen Daten anpassen muss. Zeitgleich kann bei der Anpassung die temporäre Datei gelöscht werden. Wurde der Fehler nicht behoben, wird auch die nächste Berichterstellung erneut einen Fehler ergeben. Daher muss die Datei bei einem Fehler nicht automatisch gelöscht werden. So wird bei der nächsten Durchführung, für die der Fehler behoben wurde, der Bericht erneut erstellt.

4. *Angabe von Fehlern:*
   Mit Hilfe der in dem vorherigen Punkt beschriebenen temporären Datei können Fehler für den Benutzer notiert werden. Fehlen in einer Konfigurationsdatei Daten oder fehlt eine Datei komplett, so kann eine Information in der Datei hinterlegt werden. Um dem Benutzer diese Information zukommen lassen zu können, wird die Datei nur dann gelöscht, wenn keine Fehlermeldungen hinterlegt wurden. Zudem wird zu Beginn eine Information niedergeschrieben, dass der Bericht aktuell erstellt wird, und am Ende, wenn Fehler aufgetaucht sind, dass die Berichterstellung beendet ist. Andernfalls wäre für den Benutzer nicht immer eindeutig, ob die Datei in dem Ordner liegt, weil der Bericht noch erstellt wird oder weil ein Fehler vorliegt.

5. *Löschen von nicht benötigten Kampagnen:*
   Die Kampagnen, von welchen neue Berichte erstellt werden sollen, werden lokal kopiert. Hierbei sammeln sich mit jeder neuen Kampagne neue Ordner an. Um dies zu verhindern, sollten die nicht mehr benötigten Ordner gelöscht werden. Aus diesem Grund werden die Ordner, welche in dem lokalen Ordner liegen, mit den aktuellen Ordnern in der Textdatei verglichen. Kampagnen, welche nicht aufgelistet werden, können anschließend gelöscht werden.

Neben den Sonderfällen kann die Automatisierung zudem an die aktuellen Umstände angepasst werden. Durch die COVID-19-Pandemie sind die meisten Mitarbeiter ins Home Office gewechselt. Da eine dauerhafte Verbindung zu dem Laborrechner eine gute Internetverbindung voraussetzt, wurden Alternativen entwickelt, um Testsequenzen auszuwerten. Hierbei können Sequenzen von dem Projektlaufwerk lokal gespeichert werden, um diese ohne beständige Internetverbindung bearbeiten zu können. Nach dem Abschluss einer Sequenz kann diese wieder auf das Projektlaufwerk verschoben werden. Auf dem Laufwerk besteht eine bestimmte Ordnerstruktur, in welcher die Sequenzen je nach ihrem Stand der Bearbeitung in verschiedene Ordner abgelegt werden. Dabei existiert unter anderem der Ordner "5Done", in welchem fertige Sequenzen abgelegt werden. Durch das einfache Ablegen der Sequenzen in dem Ordner und durch eine lokale Behandlung werden die Daten nicht in die Datenbank geladen. Diese Daten werden hierdurch auch nicht in den Bericht mit eingebunden. In der automatischen Berichterstellung kann die Überprüfung der neuen Daten auf die lokalen Daten erweitert werden. Hierbei wird der Ordner der Kampagne, in welchem die Sequenzen liegen, zuerst auf neue Ordner überprüft. Anschließend werden die Excel-Tabellen, welche das Ergebnis der Evaluierung sind, auf ihr Änderungsdatum geprüft. Beide Zeitstempel werden jeweils mit dem Zeitstempel der letzten Änderung in der Datenbank verglichen. Werden neue Sequenzen oder Änderungen an den Sequenzen gefunden, so wird das Programm TSEval mit Angabe der Parameter zur Verbindung mit der Datenbank ausgeführt. Durch diesen Ablauf werden neue Daten in die Datenbank geschrieben und in den Bericht eingebunden.

5.3. Umsetzung der Automatisierung

Bei der Umsetzung soll zunächst ein neues Projekt in Jenkins erstellt werden. Dieses wird mithilfe des bereits bestehenden Projektes konfiguriert. Zur Automatisierung wird hier jedoch kein Versionsverwaltungssystem eingebunden, sondern in bestimmten Zeitintervallen ein Skript ausgeführt. Dieses Skript liegt auf der virtuellen Maschine, auf welcher sich der Berichtgenerator befindet. Von Jenkins aus soll diese virtuelle Maschine als Arbeitsort verwendet werden. In dem Skript werden die benötigten Daten auf Änderungen überprüft und die Berichterstellung gestartet.

5.3.1. Konfiguration von Jenkins

In Jenkins müssen zum einen der Berichtgenerator und zum anderen die virtuelle Maschine konfiguriert werden. Dabei wird zunächst der Ort festgelegt, an welchem die Berichterstellung durchgeführt wird. Hierzu dient die virtuelle Maschine. Die weiteren Einstellungen beschreiben, dass kein Versionsverwaltungssystem verwendet wird und periodisch eine Aktion durchgeführt werden soll. Die Periodendauer wird auf 30 Minuten gesetzt. Dies wird in dem Format `*/30 * * * *` angegeben. Für die Übersichtlichkeit wird der Konsolenausgabe ein Zeitstempel hinzugefügt. Als letztes muss für den Berichtgenerator ein Kommando generiert werden, welches periodisch ausgeführt wird. Dieses Kommando ist in dem Codeabschnitt 5.1 zu sehen.

```
echo 'Begin the script'
cd /home/railsite/projects
./automatischeBerichterstellung.py
echo 'Finished'
```

Listing 5.1: Kommando, welches in Jenkins ausgeführt wird

Das Arbeitsverzeichnis wird geändert, sodass es dem Ort entspricht, an welchem das Python-Dokument liegt. Anschließend wird das Skript ausgeführt.

Für die virtuelle Maschine muss unter Jenkins eine "Node" erstellt werden. Angegeben werden dabei ein Name, eine Beschreibung, das Arbeitsverzeichnis und der Host der zu verwendenden virtuellen Maschine. Hierbei handelt es sich um die passende

IP-Adresse. Außerdem wird ein Label angegeben, welches bei dem Berichtgenerator als Arbeitsort angegeben werden kann.

5.3.2. Funktionsweise des Skripts

Im ersten Schritt werden durch das Python-Skript die Pfade der Kampagnen eingelesen und die Namen extrahiert. Hierzu liegt auf dem Projektlaufwerk eine Textdatei, in welcher die Pfade der Kampagnen zeilenweise angegeben werden. Dies übernimmt der Benutzer. So müssen nicht alle Kampagnen immer überprüft werden und der Nutzer kann auswählen, welche Berichte erstellt werden sollen. Zudem wird in der Textdatei hinter den Pfaden der Kampagnen das Datum der letzten Berichterstellung angegeben. Dies geschieht in Klammern und in dem Format `%Y-%m-%d %H:%M`. Die eingelesenen Daten werden in einem dreidimensionalen Array gespeichert. An der ersten Stelle befindet sich jeweils der Name der Kampagne, an der zweiten Stelle der entsprechende Pfad und an der dritten Stelle das dazugehörige Datum. Diese Daten können anschließend durchgegangen werden. Ist kein Datum angegeben, so wird der Bericht direkt für eine Berichterstellung gekennzeichnet. Andernfalls werden verschiedene Zeitstempel miteinander verglichen.
Der Zeitstempel der Konfigurationsdatei beschreibt den Zeitpunkt, zudem die Konfigurationsdatei das letzte Mal verändert wurde. Dieser wird mithilfe einer Funktion ermittelt und in das benötigte Format konvertiert.
Der zweite Zeitstempel, welcher ermittelt wird, beschreibt den Zeitpunkt, zu welchem die Datenbank zuletzt verändert wurde. Hierzu werden zunächst die Daten aus der Konfigurationsdatei eingelesen, welche für die Berichterstellung benötigt werden. Diese werden in einem Dictionary abgespeichert, wie in Codeausschnitt 4.5 bereits zu sehen ist. Anschließend kann eine Verbindung zu der Datenbank hergestellt werden. Der Befehl aus dem Codeausschnitt 5.2 wird auf diese Datenbank angewendet.

```
SELECT CREATE_TIME
FROM information_schema.tables
WHERE TABLE_SCHEMA = '" + parameterDict['dbname'] + "';"
```

Listing 5.2: SQL-Befehl zur Ermittlung der Erstellungsdaten

Das Ergebnis repräsentiert die Erstellungsdaten der einzelnen Sequenzen. Diese müssen ebenfalls in das richtige Format konvertiert und in einer Liste gespeichert werden. Aus den Einträgen der Liste kann das neueste Datum ermittelt werden.

Wurden alle Zeitstempel ermittelt, können diese verglichen werden. Als Erstes wird das Datum der letzten Berichterstellung mit dem Änderungsdatum der Konfigurationsdatei vergleichen. Dies geschieht mit dem Quellcode 5.3.

```
    if not timestampLastBuild == timestampDatafile:
        latest = max(timestampLastBuild, timestampDatafile)
        if latest == timestampDatafile:
            rebuild = True
```

Listing 5.3: Vergleich der Daten

Hier wird zunächst überprüft, ob die beiden zu vergleichenden Daten die gleichen sind. Ist dies nicht der Fall, kann die aktuellere Zeit von beiden ermittelt werden. Anschließend wird überprüft, ob die Konfigurationsdatei eine aktuellere Zeit ist als die der letzten Berichterstellung. In diesem Fall wird der Parameter "rebuild" auf *True* gesetzt. Somit wird der Testbericht dieser Kampagne neu erstellt. Der gleiche Code wird für das Datum der letzten Änderung an der Datenbank angewendet. Auch hier wird der Bericht markiert, wenn das Änderungsdatum der Datenbank aktueller ist als das der letzten Berichterstellung.
Als letztes wird für die betrachtete Kampagne überprüft, ob der Testbericht neu kompiliert werden soll. Dies geschieht mit dem Parameter "rebuild". Ist dieser Wert auf *True* gesetzt, wird der Pfad der Kampagne zu einer Liste hinzugefügt, in welcher alle Kampagnen gespeichert werden, von welchen der Bericht neu erstellt werden soll. Außerdem wird der Pfad zu der Kampagne zusammen mit einem aktuellen Zeitstempel in eine temporäre Textdatei geschrieben. Ist der Wert des Parameters "rebuild" auf *False* gesetzt, wird der Pfad der Kampagne der Textdatei hinzugefügt und als Datum das eingelesene Datum der letzten Berichterstellung übernommen.
Nach dem Durchlauf aller Kampagnen kann die temporäre Datei geschlossen und umbenannt werden, sodass diese die ursprüngliche Textdatei mit den Kampagnennamen überschreibt. Am Ende des Skripts erfolgt die Berichterstellung der ermittelten Kampagnen. Dabei wird die Liste mit den Pfaden durchgegangen. In der Schleife wird

der Arbeitspfad geändert, sodass der Ordner der betrachteten Kampagne das aktuelle Arbeitsverzeichnis bildet. Hier kann anschließend die Generator-Datei ausgeführt werden und das Arbeitsverzeichnis kann zurück auf den ursprünglichen Arbeitspfad geändert werden.

5.3.3. Behandlung der Sonderfälle und Fehlerbehandlung

Nach der allgemeinen Umsetzung der Automatisierung können Verbesserungen vorgenommen werden, sodass Fehler abgefangen und Sonderfälle betrachtet werden. Dabei wird sich auf die Sonderfälle und Fehler bezogen, welche in dem Konzept bereits aufgezählt wurden.

Abfangen einer fehlerhaften Berichterstellung

Für einen reibungslosen Ablauf soll der Bericht in einen lokalen Ordner kopiert werden, dort erstellt und anschließend zurück auf das Projektlaufwerk kopiert werden. Dadurch soll verhindert werden, dass Dateien von fehlerhaften Ausführungen auf dem Projektlaufwerk liegen. Außerdem wird das Python-Skript zur Berichterstellung angepasst, sodass keine Interaktion bei einer fehlerhaften Anwendung von LaTeX notwendig ist.

Im ersten Schritt wird hierzu das Skript in einem lokalen Jenkins-Ordner abgelegt. Außerdem wird in dem Jenkins-Projekt der Pfad zu diesem Ordner gewechselt und nicht mehr zu dem Ordner des Berichtgenerators. Anschließend kann zu Beginn des Skripts das aktuelle Arbeitsverzeichnis abgefragt und gespeichert werden. Danach wird zu dem Projektlaufwerk gewechselt, um die einzelnen Kampagnen durchzugehen und auf Änderungen zu überprüfen.
Am Ende des Skripts wird vor der Ausführung der Berichterstellung für die einzelnen Kampagnen zurück in das ursprüngliche Arbeitsverzeichnis gewechselt. Von dieser Stelle aus sollen die Berichte erstellt werden. Dazu werden die Namen der einzelnen Kampagnen iteriert. Für jede Kampagne wird der lokale Pfad und der Pfad auf dem Projektlaufwerk ermittelt. Mithilfe dieser kann überprüft werden, ob lokal bereits

eine Kopie der Kampagne vorliegt. Ist dies der Fall, so muss lediglich die Konfigurationsdatei von dem Projektlaufwerk kopiert und ersetzt werden, da ausschließlich an dieser Änderungen vorgenommen werden konnten. Ist noch kein passender Ordner vorhanden, so wird der ganze Ordner von dem Projektlaufwerk in den lokalen Ordner kopiert. Anschließend wird in dem lokalen Ordner die Berichterstellung gestartet. Ist diese beendet, können der erstellte Bericht und der ZIP-Ordner auf das Projektlaufwerk kopiert werden. Dabei wird der gleiche Code einmal für die PDF-Dateien und einmal für den ZIP-Ordner angewendet. Der lokale Ordner "reports" wird durchsucht nach den PDF-Dateien bzw. den ZIP-Ordnern. Dabei werden alle Namen in einer Liste gespeichert. In dieser kann basierend auf der Modifikationszeit der neueste Eintrag in der Liste ermittelt werden. Dazu wird der Befehl `latestFile = max(listOfFiles, key=os.path.getmtime)` verwendet [29]. Der Name dieser Datei wird anschließend mit dem aktuellen Zeitstempel an die Funktion übergeben, welche in dem Codeausschnitt 5.4 zu sehen ist.

```
# compare the time of the last files with the current time
def checkIfCopy(fileName, compTime):
    fileTime = getTimestampDatafile(fileName)
    if not fileTime == compTime:
        latest = max(fileTime, compTime)
        if latest == compTime:
            return False
    return True
```

Listing 5.4: Funktion zur Ermittlung, ob eine Datei kopiert werden soll

Übergeben werden der Funktion der Dateiname und die Zeit, mit welcher die Erstellungszeit verglichen werden soll. Mithilfe des Dateinamens kann die Zeit ermittelt werden, zu welcher die Datei das letzte Mal bearbeitet wurde. Diese Zeit entspricht gleichzeitig der Erstellungszeit. Für die Ermittlung wird eine Funktion verwendet, welche bereits im früheren Verlauf des Skripts zum Einsatz kommt. In dieser Funktion wird die Zeit ermittelt und in einem bestimmten Format zurückgegeben. Mithilfe der ermittelten Zeit kann überprüft werden, ob die Datei vor der aktuellen Systemzeit erstellt wurde. Dies geschieht ebenfalls mit dem Code aus dem Ausschnitt 5.3. Anstelle des Setzens eines Parameters wird in diesem Fall ein Wahrheitswert zurückgegeben. Ist die Datei älter als die aktuelle Zeit, so wird der Bericht nicht kopiert. Andernfalls

wird die Datei auf das Projektlaufwerk kopiert. Dadurch steht hier zu jeder Zeit die aktuellste Version des Berichts zur Verfügung.

In der Python-Datei zu der Berichterstellung wird der Aufruf von Python angepasst. Aktuell wird die PDF mithilfe des in Codeausschnitt 5.5 zu sehenden Befehls erstellt.

```
call(["pdflatex", "berichtB3"])
```

Listing 5.5: Ursprünglicher Befehl zum Erstellen einer PDF-Datei

Hierbei kommt es zu Problemen, wenn falsche Daten in der Konfigurationsdatei angegeben werden oder ein Parameter keinen Wert zugewiesen bekommt. In diesem Fall muss die Berichterstellung manuell abgebrochen werden. Aus diesem Grund muss der Befehl erweitert werden. Diese Erweiterung ist in dem Codeausschnitt 5.6 zu sehen.

```
call(["pdflatex", "-halt-on-error", "berichtB3"])
```

Listing 5.6: Erweiterter Befehl zum Erstellen einer PDF-Datei in Bezug auf Fehler

Hier wird die Flag "-halt-on-error" angegeben. Durch diesen Parameter wird der Befehl abgebrochen, sobald der erste Fehler entsteht. Allerdings können hierbei weiterhin Eingaben gefordert werden, wenn beispielsweise eine Datei nicht gefunden werden kann. Diese Fehler müssen auf eine andere Art abgefangen werden. Der Parameter "-halt-on-error" wird vor allem für Automatisierungszwecke verwendet. [32]

Threading für mehrere Berichte

Wurden an mehreren Kampagnen zeitgleich Änderungen vorgenommen, so können diese durch Threading parallel behandelt werden. Hierzu wird zunächst die Generator-Datei angepasst, indem bei dem Aufruf von LaTeX eine weitere Flag gesetzt wird. Diese ist in dem Codeausschnitt 5.7 zu sehen.

```
call(["pdflatex", "-halt-on-error", "-interaction=batchmode",
    "berichtB3"])
```

Listing 5.7: Erweiterter Befehl zum Erstellen einer PDF-Datei in Bezug auf die Ausgabe

Durch den Parameter "-interaction=batchmode" werden alle Ausgaben außer bestimmte beschreibende Zeilen bei der Ausführung von LaTeX unterdrückt. Hierzu zählt zum Beispiel die Version von LaTeX, welche für das Kompilieren verwendet wird. Mit Hilfe dieser Flag wird die Ausgabe auf der Konsole übersichtlicher und es kann besser nachvollzogen werden, welche Kampagnen betrachtet wurden. [31]

Anschließend wird das Skript für Jenkins angepasst. Aktuell werden die zu betrachtenden Kampagnen in einer Liste gespeichert und diese wird durchgegangen, um die einzelnen Berichte zu erstellen. Der Inhalt dieser Schleife wird nun in eine Funktion mit dem Namen "createReport" ausgelagert. Als Übergabeparameter erhält die Funktion den Namen der Kampagne. Die Funktion wird innerhalb der Erstellung der Threads aufgerufen. Dies ist in dem Codeausschnitt 5.8 zu sehen.

```
numCampaigns = len(campaignsToRebuild)
if numCampaigns > 0:
    threads = list()
    for campaignPath in campaignsToRebuild:
        x = threading.Thread(target = createReport, args = (
            campaignPath,))
        threads.append(x)
        x.start()
    for thread in threads:
        thread.join()
```

Listing 5.8: Anwenden von Threads

Zuerst wird die Anzahl der zu erstellenden Berichte ermittelt, um sicherzustellen, dass mindestens ein Bericht erstellt werden soll. Anschließend werden die Kampagnen durchgegangen und pro Kampagne wird ein Thread erstellt. Bei der Erstellung wird die Funktion "createReport" aufgerufen und als Übergabeparameter wird der Pfad der Kampagne angegeben. Der erstellte Thread wird dann zu einer Liste hinzugefügt, sodass später wieder auf diesen zugegriffen werden kann. Anschließend

wird der Thread gestartet. Im Folgenden werden die erstellten Threads nochmals durchgegangen. Hierbei wird der Befehl `thread.join()` ausgeführt. Dadurch wartet ein Thread auf den anderen, bis dieser ebenfalls beendet wird. Konkret bedeutet das in diesem Fall, dass der Haupt-Thread von dem Programm nicht beendet wird, solange noch Berichte erstellt werden. Das Programm wird erst dann beendet, wenn alle Berichte fertig erstellt und auf das Laufwerk kopiert wurden. [7]

Handhabung von Änderungen in kurzer Zeit

Bei der Berichterstellung muss der wechselseitige Ausschluss beachtet werden. Hierbei wird anstelle von Semaphoren, welche beim Threading verwendet werden können, eine temporäre Datei erstellt. Mit Hilfe von dieser kann eine Interaktion zwischen verschiedenen Prozessen umgesetzt werden. So kann verhindert werden, dass mehrerer Berichte von der gleichen Kampagne erstellt werden. Hierzu wird der Code für das Jenkins-Skript angepasst. Bei dem Einlesen der Kampagnenamen wird, wenn keine Zeit in der Textdatei für die letzte Erstellung angegeben ist, überprüft, ob die temporäre Datei bereits existiert. Ist dies der Fall, so wird diese gelöscht und der Bericht kann neu erstellt werden. Ist eine Zeit angegeben und die temporäre Datei existiert, so wird eine Information ausgegeben, dass der Bericht bereits erstellt wird. Zudem wird an dieser Stelle die Kampagne mit dem übernommenen Zeitstempel in die neue Textdatei geschrieben und die nächste Kampagne wird betrachtet.
Im weiteren Verlauf muss der Code bei der Erstellung der Testberichte angepasst werden. Zu Beginn wird in der for-Schleife, in welcher die Kampagnen zur Berichterstellung durchgegangen werden, eine temporäre Datei mit dem Namen "temp.txt" erstellt. Diese befindet sich in dem Ordner der Kampagne auf dem Projektlaufwerk. Erstellt wird sie mit dem Befehl in dem Codeausschnitt 5.9.

```
tmp = open(campaignDirectory+'/'+campaignPath+'/temp.txt','w')
tmp.close()
```

Listing 5.9: Erstellen der temporären Textdatei

Durch das Öffnen einer Textdatei mit dem Buchstaben "w" als Argument wird diese Datei erstellt, wenn sie nicht vorhanden ist [12]. Um später ggf. nochmals auf diese

Datei zugreifen zu können, muss sie anschließend wieder geschlossen werden. Nach der Berichterstellung und nach dem Kopieren der fertigen Daten kann die Datei wieder gelöscht werden. So kann der Bericht bei dem nächsten Durchlauf neu erstellt werden, wenn Änderungen an den Daten vorgenommen wurden.

Tritt ein Fehler bei der Berichterstellung auf und die Datei wird nicht gelöscht, so muss der Fehler zunächst behoben werden. Anschließend kann die temporäre Datei händisch gelöscht werden, um bei dem nächsten Durchgang die Berichterstellung erneut zu versuchen.

Fehlerangabe für den Benutzer

Als Unterstützung für den Anwender wird die temporäre Datei zur Hilfe gezogen. Hierbei sollen Fehlermeldungen in dieser Datei dokumentiert werden, sodass der Benutzer eine einfache Möglichkeit hat, das Problem bei der Berichterstellung zu erkennen.
Hierzu wird zunächst das Erstellen der Datei angepasst. Dabei werden bereits bei dem Einlesen der Kampagnennamen Änderungen vorgenommen. Ist in der Textdatei keine Zeit angegeben, es existiert aber bereits eine temporäre Datei, so wird von dieser der Inhalt gelöscht, indem die Datei zum Schreiben geöffnet wird. Dabei wird der Inhalt der existierenden Datei überschrieben. Existiert die temporäre Datei noch nicht und in der Textdatei ist ein Datum angegeben, müssen die Änderungszeiten betrachtet werden. Hier wird zunächst die temporäre Datei neu erstellt. Anschließend wird sie bei der Ermittlung von den Änderungszeiten an die verwendeten Funktionen übergeben. Dadurch können Fehlermeldungen generiert werden, wenn keine Konfigurationsdatei vorhanden ist oder wenn Daten fehlen, um eine Verbindung mit der Datenbank aufbauen zu können. Danach wird die Datei wieder geschlossen. Wurde keine Änderung an der Kampagne festgestellt, kann die temporäre Datei direkt wieder gelöscht werden.

Vor dem Bilden der Testberichte wird die temporäre Datei erneut für die aktuell betrachtete Kampagne geöffnet. Beim Aufrufen des Ordners der Kampagne wird zunächst überprüft, ob der benötigte Ordner existiert. Ist dies nicht der Fall, wird

eine Information in die temporäre Datei geschrieben. Bei jeder Erstellung des Berichts wird in jedem Fall eine Information in die Textdatei geschrieben, welche den Benutzer darauf hinweist, dass der Bericht zur Zeit noch erstellt wird. Anschließend wird die Datei wieder geschlossen.

Weiter wird in der zusätzlichen Datei mit den ausgelagerten Funktionen eine weitere Funktion erstellt. Diese ist in dem Codeausschnitt 5.10 zu sehen.

```
def writeToTxt(text):
    print(text)
    tempLogFile = './tempLocal.txt'

    # write the error to the text file
    tmp = open(tempLogFile, 'a')
    tmp.write(text + '\n')
    tmp.close()
```

Listing 5.10: Funktion zum Schreiben von Fehlermeldungen in eine Textdatei

Als Übergabeparameter wird von der Funktion ein Text entgegengenommen. Dieser wird in dem ersten Schritt auf der Konsole ausgegeben. Im weiteren Verlauf wird der Name der Datei angegeben, welche sich in dem lokalen Ordner befindet. Diese wird in dem Generator als erster Schritt erzeugt. Es wird nicht direkt in die Datei auf dem Laufwerk geschrieben, da es in diesem Fall Probleme mit der Pfadangabe gibt. Diese kann zwischen den einzelnen Kampagnen variieren, weshalb eine zusätzliche lokale Datei eine einfache Alternative bildet. Anschließend wird die Datei mit dem Argument "a" geöffnet, um ausschließlich Inhalte ergänzen zu können. Dies geschieht, indem der übergebene Text in die Datei geschrieben und mit einem Zeilenumbruch ergänzt wird. Am Ende wird die Datei wieder geschlossen.
Verwendet wird die Funktion zum Schreiben der Fehlermeldungen in eine Textdatei in den beiden Funktionen zum Einlesen der Daten. Bei beiden ist eine Ausgabe eingebunden, wenn ein Parameter nicht angegeben wurde. Diese Fehlermeldung wird in die Funktion gegeben. Zudem wird die Funktion in dem Generator eingebunden. Hier wird die Information, wenn eine Frage nicht korrekt beantwortet wurde, an die Funktion übergeben. Außerdem wird bei dem Einlesen der Konfigurationsdatei die Möglichkeit hinzugefügt, dass diese nicht in dem Ordner vorhanden ist. Auch in

diesem Fall wird ein kurzer Text in die temporäre Datei geschrieben. Des Weiteren werden Informationen aufgenommen, die auf Fehler in den Sequenzen hinweisen.

In dem Skript, welches in Jenkins eingebunden ist, wird zudem das Löschen der Datei angepasst. Zuerst wird die lokale temporäre Datei eingelesen und direkt gelöscht. Der eingelesene Inhalt wird, wenn die Datei nicht leer war, in der temporären Datei auf dem Laufwerk ergänzt.
Am Ende der Berichterstellung wird die Datei nochmal mit dem Argument "a+" geöffnet. Hierdurch kann der Inhalt der Datei eingelesen oder ergänzt werden. In diesem Fall wird zunächst der Befehl `tmp.seek(0)` ausgeführt, um die Datei komplett einlesen zu können und den Cursor an den Anfang zu setzen. Anschließend kann der Inhalt eingelesen und überprüft werden. Entspricht der Inhalt dem Satz, welcher zu Beginn in das Dokument geschrieben wurde, so kann die Datei gelöscht werden. Sind andere Inhalte enthalten, so bleibt die Datei bestehen, um dem Benutzer den Fehler mitteilen zu können. Zudem wird in diesem Fall die Datei nochmals mit einer Information ergänzt, dass die Berichterstellung mit einem Fehler abgeschlossen wurde. Der letzte Schritt ist anschließend das Schließen der Datei.

Löschen von nicht benötigten Kampagnen

Zur Überprüfung, welche Ordner lokal gelöscht werden können, werden zunächst die Namen aller lokalen Ordner in einer Liste gespeichert. Dieser Vorgang ist in dem Codeausschnitt 5.11 zu sehen.

```
    for name in os.listdir('.'):
        if os.path.isdir(os.path.join('.', name)):
            listOfLocalCampaigns.append(name)
```

Listing 5.11: Einlesen der lokalen Ordner

In der for-Schleife werden alle Inhalte des Ordners durchgegangen. Jedes Element wird darauf überprüft, ob es sich dabei um einen Ordner handelt. Ist dies der Fall, so wird der Name des Ordners zu der erstellten Liste hinzugefügt.
Der Ordner *BerichtVorlagen* wird anschließend wieder aus der Liste gelöscht, da es sich hierbei um benötigte Dateien handelt, welche von allen Kampagnen für die

Erstellung des Berichts benötigt werden.
Werden die einzelnen Kampagnen in der Textdatei eingelesen und betrachtet, wird hier zudem die Liste der zu löschenden Kampagnen angepasst. Ist ein Name einer Kampagne in der Textdatei vorhanden, wird dieser aus der erstellten Liste gelöscht. So besteht die Liste schlussendlich nur aus Kampagnen, welche nicht mehr betrachtet werden, aber lokal bereits gespeichert wurden. Diese Liste wird am Ende des Skripts vor der Erstellung der einzelnen Testberichte durchgegangen und die jeweiligen Ordner werden mit dem Befehl `shutil.rmtree(campaign)` zusammen mit ihrem Inhalt gelöscht. Durch diese Erweiterung wird der lokale Ordner regelmäßig überprüft und aufgeräumt.

5.3.4. Erweiterung der Automatisierung für lokale Änderungen

Um bei der Berichterstellung auch auf lokale Änderungen reagieren zu können, müssen die Ordner der einzelnen Sequenzen auf Änderungen überprüft werden. Hierzu werden zunächst die Namen aller Sequenzen in dem Ordner "5Done" der entsprechenden Kampagne eingelesen. Bereits hierzu ist die richtige Struktur der Ordner relevant. Diese ist in der Abbildung 5.4 zu sehen. Die Struktur wird jedoch auf die hier relevanten Ordner beschränkt. In den einzelnen Ebenen sind weitere Ordner zu finden, welche beispielsweise Daten und Skripte enthalten, die für die automatische Berichterstellung und das Schreiben der Daten in die Datenbank nicht benötigt werden.
Diese Struktur muss in dem Aufbau jeder Kampagne eingehalten werden. So können Daten einfach kopiert oder überprüft werden. In dem Ordner "5Done" liegen alle Ordner der einzelnen fertigen Sequenzen. In der Abbildung 5.4 sind beispielsweise drei Sequenzen aufgezeigt, welche in diesem Beispiel einfache Namen bekommen haben. Die eigentliche Benennung erfolgt durch eine Zahlenfolge, durch welche jede Sequenz eindeutig identifizierbar ist.
Nach dem Einlesen der einzelnen Kampagnen werden die Ordner nacheinander überprüft. Hierzu wird in dem ersten Schritt der Zeitstempel des Ordners ermittelt. Ist dieser aktueller als der Zeitstempel der Datenbank, so wird der Name der Kampagne zu einer Liste hinzugefügt. Andernfalls werden die beiden Excel-Dateien überprüft,

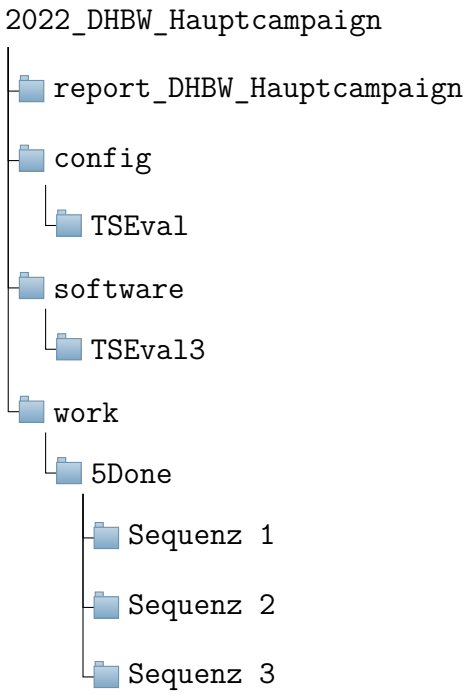


Abbildung 5.4.: Ordnerstruktur der einzelnen Kampagnen

welche bei der Evaluierung erstellt und bearbeitet werden. In der Abbildung 5.5 ist die Ordnerstruktur einer einzelnen Sequenz zu sehen. Diese ist beschränkt auf die Teile, welche für den Berichtgenerator relevant sind.

Die beiden Excel-Dateien, welche auf ihr Änderungsdatum überprüft werden, sind in dem Unterordner *evaluation* zu finden. Wurde von den beiden Dateien eine nach der letzten Änderung der Datenbank bearbeitet, wird auch hier der Name der Sequenz zu der Liste hinzugefügt.
Anschließend können die einzelnen Sequenzen in die Datenbank übertragen werden. Hierzu werden in dem ersten Schritt die Software und die Konfigurationsdaten aus dem Kampagnenordner auf die virtuelle Maschine kopiert. Die Ordnerstruktur, welche in der Abbildung 5.4 zu sehen ist, wird in dem Ordner "tempSequences" nachgebildet, um bei dem auszuführenden Skript die richtigen Pfadangaben verwenden zu können. Der Unterschied liegt darin, dass der Ordner für den Testbericht auf der virtuellen Maschine eine Ebene weiter oben liegt, da dieser nicht in das Laden der einzelnen Sequenzen involviert ist. Wurden die Ordner "config" und "software" mit

Abbildung 5.5.: Ordnerstruktur der einzelnen Sequenzen

dem jeweiligen Unterordner "TSEval" bzw. "TSEval3" in den temporären Sequenzenordner kopiert, können die einzelnen Sequenzen betrachtet werden. Hierzu werden die Sequenznamen, welche in einer Liste zusammengefasst wurden, nacheinander durchgegangen. Dabei wird der ganze Ordner der betrachteten Sequenz in den Ordner "5Done" auf der virtuellen Maschine kopiert und das Arbeitsverzeichnis wird in den Unterordner "TSEval" gewechselt. Von hier aus wird TSEval, welches kompiliert in dem Ordner "software" abgespeichert wurde, ausgeführt. Bei dem Aufruf werden die Datei "TSEvalLabRun.conf" und die Parameter für die Verbindung mit der Datenbank übergeben. Zudem handelt es sich um den Befehl für die automatisierte Ausführung im Labor, welcher an dieser Stelle ausgeführt wird. Der Unterschied zu dem einfachen Aufruf liegt in den Parametern, welche zusätzlich übergeben werden. Durch zwei Erweiterungen wird die Software nach Ausführung automatisch wieder geschlossen und bei dem Auftauchen eines Fehlers wird das Programm direkt beendet. So kann es zu keinen Fehlern kommen, durch die sich die Automatisierung aufhängen würde.

Durch die Ausführung von TSEval werden die Daten in die Datenbank eingebunden. Um zu prüfen, ob hierbei ein Fehler aufgetreten ist, wird der Inhalt der Datenbank

nach der Durchführung nochmals eingelesen. Enthält die Liste mit den Daten aus der Datenbank den Namen der aktuellen Sequenz, so ist kein Fehler aufgetreten. Andernfalls wird in der temporären Datei ein Hinweis angefügt. Außerdem wird, verbunden mit der Überprüfung, die alte Version der Sequenz aus der Datenbank gelöscht. Hierzu wird zunächst eine Liste angelegt, welche den hinteren Teil der Tabellennamen in der Datenbank beinhaltet. Dabei werden nur die Tabellen betrachtet, welche den Sequenznamen in ihrem Namen haben. Bei einer erfolgreichen Durchführung liegen maximal zwei Tabellen der gleichen Sequenz in der Datenbank. Von diesen beiden wird überprüft, welcher Zeitstempel der aktuellere ist. Hierzu wird der hintere Teil des Namens verwendet, da es sich hierbei um einen Zeitstempel handelt. Die Tabelle, welche den entsprechenden Sequenznamen, aber einen anderen Zeitstempel in dem Tabellennamen enthält, wird aus der Datenbank entfernt. So befindet sich am Ende der Berichterstellung von jeder Sequenz genau eine Version in der Datenbank.
Direkt im Anschluss an die Überprüfung und das Löschen der alten Version wird der Ordner der Sequenz wieder gelöscht. Nach dem Durchgehen aller Sequenzen werden zudem die Ordner ”software”, ”config” und die leere Ordnerstruktur ”work” wieder gelöscht.

6. Umsetzung der Konfigurationsdatei in Form einer interaktiven PDF

Für eine benutzerfreundlichere Anwendung wird neben der Textdatei eine interaktive PDF erstellt. Hierbei muss der Benutzer nicht auf die Formatierung achten, da einfache Textfelder in der Datei enthalten sind, welche ausgefüllt werden können. Dabei wurde sich für das Format einer PDF entschieden, da diese einfach verschoben werden kann. Die Datei muss flexibel von dem Projektlaufwerk auf die virtuelle Maschine verschoben und zudem einfach kopiert werden können, um für die einzelnen Kampagnen jeweils einen einzelnen Ordner erstellen zu können.
Zudem werden keine Animationen oder Ähnliches in der PDF benötigt. Daher ist diese Lösung ausreichend. Eine andere Möglichkeit wäre die Verwendung einer HTML-Datei. Dafür spricht, dass hierbei Animationen, Videos oder Ähnliches eingebunden werden können und daher im Vergleich die interaktive PDF nicht mehr bevorzugt wird [13]. Für den Berichtgenerator reichen jedoch die angebotenen Möglichkeiten einer interaktiven PDF aus. Hier müssen lediglich Daten eingetragen und wieder ausgelesen werden. Diese Form der Konfigurationsdatei ist außerdem leicht verständlich und bei allen Benutzern bekannt.

Im Folgenden wird zunächst erläutert, wie die interaktive PDF erstellt werden kann. Anschließend werden die Daten, welche in der PDF eingetragen werden, mithilfe von Python eingelesen und der Berichtgenerator wird so modifiziert, dass die Daten eingebunden werden können. Zum Schluss muss zudem noch das Jenkins-Skript angepasst werden. Hier muss überprüft werden, ob die Konfigurationsdatei als

Textdatei oder als PDF in dem betrachteten Ordner vorliegt. Entsprechend kann die Datei dann auf Änderungen überprüft und benötigte Daten eingelesen werden.

6.1. Erstellen einer interaktiven PDF

Zum Erstellen einer interaktiven PDF kann LibreOffice verwendet werden. Dabei muss die Konfigurationsdatei als "data.odt" abgespeichert werden. In diesem Format können auch interaktive Elemente der Datei gespeichert werden, ohne die Datei direkt in eine PDF konvertieren zu müssen.
In dem ersten Schritt wird die Grundlage der Datei gebildet. Es werden alle Textelemente eingebunden, welche nicht interaktiv sind. Darunter zählen die Namen der einzelnen Parameter, aber auch erklärende Texte, welche Parameter erläutern, deren Name nicht selbsterklärend ist. Zudem wird zu Beginn eines Abschnitts eine Erklärung angegeben, welche Parameter als Nächstes folgen. Die einzelnen Abschnitte sind die Folgenden:

1. *Parameter für die Erstellung*
   In diesem Abschnitt werden zum einen der Name für die verschiedenen zu erstellenden Dateien und die Angaben für die Verbindung zu der Datenbank abgefragt und zum anderen gibt es drei Fragen, welche mit "Ja" oder "Nein" beantwortet werden müssen. Dabei handelt es sich um die drei Fragen, welche für die automatische Berichterstellung ebenfalls in die Konfigurationsdatei eingebunden wurden.

2. *Parameter zum Einbinden in den Bericht*
   In diesem Abschnitt sind alle Parameter zu finden, welche in den Bericht eingebunden werden wie die Daten zu dem Kunden oder zu dem Labor.

3. *Freitexte in dem Bericht*
   Der letzte Abschnitt umfasst alle Freitexte, welche in den Bericht eingebunden werden können. Das sind die Modifikationshistorie und Erweiterungen der Kapitel, welche ergänzt werden müssen, sobald eine Delta-Kampagne durchgeführt wird.

Ist die Grundlage geschaffen, können die interaktiven Elemente eingebunden werden. Dazu muss zunächst die "Form Controls toolbar" geöffnet werden. In dieser befinden sich die Elemente, welche in das Dokument eingebunden werden sollen. Für die einfachen Parameter werden Textfelder eingebunden. Diese bekommen einen Namen, durch welches das Feld eindeutig identifizierbar ist. Zudem muss der Parameter "Anchor" auf "To Paragraph" gesetzt werden, damit das Feld an den jeweiligen Parameter gebunden wird. So kann das Dokument weiter bearbeitet werden und es können an einer früheren Stelle Zeilen eingefügt werden, ohne dass die Textfelder anschließend neu angeordnet werden müssen. Zudem wird der Parameter "Text type" auf "Single-line" gesetzt, da nur kurze Antworten erwartet werden. [28]
Für Fragen, welche mit "Ja" oder "Nein" beantwortet werden sollen, werden "Option Buttons" verwendet. Hierbei müssen zusammengehörige Button den gleichen Namen bekommen. So wird festgelegt, dass nur ein Button ausgewählt werden kann. Für den Parameter "Label" wird der Text angegeben, welcher neben dem Button in dem Dokument erscheinen soll. Zudem muss bei dem Parameter "Default status" angegeben werden, ob der Button in dem ersten Zustand aktiviert sein soll oder nicht. [28]
Die dritte Art von interaktiven Elementen sind die Textfelder in dem dritten Abschnitt. Dabei wird ebenfalls eine "Text Box" eingebunden, welche ähnlich zu den einfachen Textfeldern konfiguriert ist. Der Unterschied besteht in dem "Text type", welcher hierbei auf "Multi-line" gesetzt wird. Dies ist wichtig, da teilweise größere Textblöcke eingebunden werden müssen. Als Erweiterung wird eine vertikale Scrollleiste eingebunden. Dadurch kann der einzubindende Text länger als das Textfeld sein.

In dem Abschnitt A.2 sind Auszüge aus der Konfigurationsdatei zu sehen. Ein erster erklärender Text und der erste Abschnitt mit den Parametern zu der Berichterstellung sind in der Abbildung A.3 zu sehen. In der Abbildung A.4 befindet sich der Abschnitt, welcher Parameter beinhaltet, die in den Bericht eingebunden werden sollen. Die letzten beiden Abbildungen A.5 und A.6 zeigen die Freitexte und die Frage, ob es sich bei der Kampagne um eine Delta-Kampagne handelt.

6.2. Einlesen der Daten und Einbinden in den Berichtgenerator

Für das Einlesen der Textfelder in der PDF-Datei wird das Modul PyPDF2 verwendet. Dieses muss zunächst auf der virtuellen Maschine installiert werden. Da auf dieser Maschine keine Internetverbindung besteht, muss auf dem lokalen Rechner die passende **Wh**eel (WHL)-Datei heruntergeladen werden. Hierbei handelt es sich um ein "Python Wheel Package", was "alle Dateien für eine Python-Installation und Metadaten"[25] enthält. Heruntergeladen wird die Datei auf dem lokalen Rechner mit dem Befehl `pip download PyPDF2 -d .`. Diese Datei kann auf das Projektlaufwerk geladen und von hier aus installiert werden. Die Installation geschieht, indem der Befehl `pip3 install * -f ./ --no-index --no-deps` ausgeführt wird. Durch das "*" werden alle Dateien in dem Ordner auf der virtuellen Maschine installiert. Soll nur eine bestimmte Datei installiert werden, so muss das "*" durch den Namen der entsprechenden WHL-Datei ersetzt werden. [30]
Zur Anwendung des Paketes muss in dem Skript als erstes die Zeile `#!/usr/bin/python3` angegeben werden, da ansonsten auf die installierte Version 2 zugegriffen wird. So kann das Modul PyPDF2 eingebunden und verwendet werden. Allerdings führt dies zu Versionsproblemen mit anderen eingebundenen Bibliotheken. Daher muss hier der gleiche Ablauf nochmals für die Pakete, welche nicht in der Python-Version 3 gefunden werden können, durchgeführt werden.

Zum Einlesen der Konfigurationsdatei wird ein neues Python-Skript erstellt, welches Funktionen enthält, die von dem Hauptdokument aus aufgerufen werden können. Hierdurch wird das Programm modular erstellt und kann einfach auch für die Konfigurationsdatei in Form einer Textdatei angepasst werden. Hierzu werden zunächst die einzelnen Funktionen erstellt. Diese können im Anschluss in das Hauptskript eingebunden werden. Der nächste Schritt besteht in der Anpassung der Version mit der Textdatei, um diese ebenfalls modular vorzubereiten.

6.2.1. Erstellen des Python-Skripts zum Einlesen der Daten

In dem zusätzlichen Python-Skript werden zunächst drei Listen erstellt, welche alle Parameter der Fragen in der Konfigurationsdatei, alle Parameter für den Bericht und alle Kapitel-Überschriften für die Freitexte enthält. Diese werden im weiteren Verlauf benötigt, um bestimmte Einträge anpassen zu können und um auf bestimmte Einträge in dem Dictionary zugreifen zu können, welches im Folgenden erstellt wird. Für alle drei Listen werden Funktionen erstellt, welche diese zurückgeben, um von dem Hauptdokument auf diese zugreifen zu können.
Außerdem existiert in dem Dokument die Funktion `readPdf()`. Diese ist in dem Codeausschnitt 6.1 zu sehen.

```
def readPdf():
    # read file in complete text
    pdfFile = open(parameterFile, 'rb')
    readPdf = PyPDF2.PdfFileReader(pdfFile)
    fields = readPdf.getFields()

    for fieldName, value in fields.items():
        fieldValue = value.get('/V', None)
        if fieldName == "":
            print ('Parameter ' + fieldName + ' not found.')
        elif fieldName in questions:
            if fieldValue == '/1':
                fieldValue = 'yes'
            elif fieldValue == '/2':
                fieldValue = 'no'
        elif fieldName in parameterReport:
            fieldValue = fieldValue.replace('_', r'\_')
        dictionary[fieldName] = fieldValue

    dictionary['modificationHistory'] = dictionary['
       modificationHistory'].replace(':', '&')
    dictionary['modificationHistory'] = dictionary['
       modificationHistory'].replace('\n', '\\\\')

    return dictionary
```

Quelle: [27]

Abbildung 6.1.: Funktion zum Extrahieren der Daten aus der PDF

In dem ersten Schritt wird die PDF-Datei geöffnet und mit dem Modul *PyPDF2* eingelesen. Aus den eingelesenen Daten können die interaktiven Felder extrahiert und durchgegangen werden. Dabei ist der Wert mit dem Parameter "\V" in den Daten aufgezeichnet. Ist der betrachtete Parameter in der Liste mit den Fragen enthalten, so muss der Eintrag konvertiert werden. Dabei steht eine 1 für ein "Ja" und eine 2 für ein "Nein". Außerdem werden die Einträge für den Bericht ebenfalls, so wie in Abschnitt 4.3 bereits erläutert, behandelt. Die Modifikationshistorie wird ebenfalls, wie in Kapitel 4.4 beschrieben, angepasst.

6.2.2. Anpassen des Hauptdokuments

In dem Hauptdokument werden zunächst die benötigten Funktionen aus dem zusätzlichen Dokument eingebunden und es werden ein Dictionary und zwei Listen initialisiert. Die bereits bestehenden Dictionarys, welche ursprünglich an dieser Stelle erstellt wurden, werden zusammen mit der Funktion zum Schreiben von Daten in ein Dictionary gelöscht. Anstelle des Einlesens der Textdatei und dem Aufteilen der Daten in verschiedene Dictionarys wird die Funktion aus dem Codeausschnitt 6.1 aufgerufen. Anschließend werden die Daten für den Bericht, so wie in der ehemaligen Version, als Kommandos in eine LaTeX-Datei geschrieben. Hierzu wird die Liste mit den Parametern für den Bericht aus dem anderen Dokument aufgerufen und alle Parameter aus dieser werden in Kommandos umformuliert. Bei dem Schreiben der Kapitelinhalte in einzelne LaTeX-Dokumente werden zwei Änderungen angepasst. Zum einen wurde in der Konfigurationsdatei eine Frage eingebunden, ob es sich bei der Kampagne um eine Delta-Kampagne handelt. Ist dies der Fall, so wird eine Variable auf *True* gesetzt, andernfalls auf *False*. Mit Hilfe der Liste mit den Überschriften der Kapitel werden anschließend die einzelnen Kapitel durchgegangen. Ist die Variable auf *True* gesetzt, wird der Inhalt aus dem entsprechenden Textfeld in das Dokument geschrieben. Ist dies nicht der Fall, werden leere Dokumente erstellt. So entstehen keine Fehler bei der Erstellung des Berichts, da hier die jeweiligen Abschnitte immer eingebunden werden.

Durch das Auslagern von bestimmten Funktionen in eine andere Datei wird bei der Ausführung des Skripts ein Ordner namens "__pycache__" erstellt. Dieser kann

am Ende des Skripts zusammen mit den anderen temporären Daten wieder gelöscht werden. Dazu wird die Zeile `call(["rm", "-r", "__pycache__"])` an dem Ende des Hauptdokuments ergänzt.
Weitere Anpassungen an dem Skript betreffen hauptsächlich die Umbenennungen der aufgerufenen Dictionaries, welche von drei verschiedenen auf ein gemeinsames reduziert wurden.

6.2.3. Anpassen des Einlesens der Textdatei

Zur Modularisierung und einfachen Erweiterbarkeit wurde sich dazu entschieden, das Einlesen der Textdatei anzupassen, sodass das gleiche Hauptdokument verwendet werden kann. Dabei muss lediglich eine Funktion in dem Skript, welches die ausgelagerten Funktionen enthält, angepasst werden. Der Vorteil an dieser Anpassung liegt in der guten Wartbarkeit der Software. Das Hauptdokument kann so weiterentwickelt werden, ohne das Versionsprobleme entstehen können. Es muss nur ein Skript angepasst werden, welches anschließend in den anderen Ordner kopiert werden kann.

Für die Anpassung bei dem Einlesen der Textdatei wird der Teil des Hauptdokuments zunächst kopiert, welcher die Textdatei einliest. Dieser Teil muss so angepasst werden, dass anstelle von verschiedenen Dictionaries nur eins erstellt wird, in welchem alle wichtigen Informationen enthalten sind. Der grundlegende Aufbau der zusätzlichen Datei ist der gleiche wie bei dem Einlesen der PDF. Inhaltlich wird lediglich die Funktion zum Einlesen angepasst. Dabei werden die Schritte aus der ursprünglichen Version zum Einlesen der Textdatei angewendet. Statt der Funktion zum Einbinden von Daten in ein gegebenes Dictionary wird hier jedoch alles zusammengeführt. Der ganze Code ist in dem Codeausschnitt B.1 zu sehen. In dem ersten Schritt werden die Daten an dem Seperator "#####" in einzelne Listeninhalte aufgeteilt. Diese Liste wird anschließend durchgegangen und die Einträge werden jeweils an dem ersten Doppelpunkt geteilt. Für die Fehlerbehandlung wird überprüft, ob nach dem Teilen zwei Teile vorhanden sind. Dies ist notwendig, falls eine leere Zeile betrachtet wird. Der erste Teil wird im Folgenden als Header abgespeichert und der zweite als Body. Ist der Header in der Liste mit den Kapitelüberschriften enthalten, so wird die erste

Leerzeile abgeschnitten und der Inhalt wird als Eintrag in dem Dictionary ergänzt. Handelt es sich bei dem betrachteten Eintrag um die Änderungshistorie, muss der Inhalt der Textdatei wie in Kapitel 4.4 angepasst werden. Anschließend kann die Historie ebenfalls dem Dictionary hinzugefügt werden. Die dritte Möglichkeit besteht darin, dass der Abschnitt die einzelnen Parameter zur Berichterstellung und zum Einbinden in den Bericht enthält. Ist dies der Fall, so muss der Body nochmals zeilenweise geteilt werden. Anschließend können die einzelnen Zeilen durchgegangen und ebenfalls an dem ersten Doppelpunkt getrennt werden. Der erste Teil beschreibt nun den Parameter und der zweite den Eintrag für diesen. Sind zwei Teile vorhanden, so kann zunächst überprüft werden, ob ein Eintrag existiert oder ob dieser Teil leer ist. Ist kein Eintrag in der Textdatei vorhanden, wird eine Information ausgegeben, dass der Parameter nicht gefunden wurde. Andernfalls wird der Parameter in das Dictionary aufgenommen.

6.2.4. Kombinieren der beiden erstellten Berichtgeneratoren

Um nicht zwei verschiedene Versionen des Berichtgenerators verwalten zu müssen, können diese beiden kombiniert werden. Hierzu werden die Dateien "readData.py" kombiniert, indem ein gemeinsames Skript erstellt wird, in welchem beide Funktionen zum Einlesen der Konfigurationsdateien enthalten sind. Anschließend wird in dem Ordner des Berichtgenerators die Konfigurationsdatei in Form einer Textdatei und in Form einer interaktiven PDF abgelegt. So kann der Benutzer selbst entscheiden, welche von beiden er verwenden möchte. Schlussendlich muss nur die "generatorB3.py" angepasst werden. Hier werden zu Beginn die beiden Funktionen zum Einlesen der Daten eingebunden anstatt der einen gleich benannten Funktion. Des Weiteren müssen diese Funktionen anschließend aufgerufen werden. Der Code hierzu ist in dem Codeabschnitt 6.1 zu sehen.

```
if os.path.exists('data.pdf'):
    dataDictionary = readConfPdf()
elif os.path.exists('data.txt'):
    dataDictionary = readConfTxt()
```

Listing 6.1: Einlesen der Daten mit Unterscheidung der Konfiguration

Hierbei wird zunächst überprüft, ob die Konfigurationsdatei in Form einer PDF vorliegt. Ist dies des Fall, so wird die entsprechende Funktion aufgerufen und so die Daten eingelesen. Andernfalls wird überprüft, ob die Datei in Form der Textdatei vorliegt. Auch hier wird dann die entsprechende Funktion aufgerufen und die Daten eingelesen. Durch die Reihenfolge der Abfragen wird deutlich, dass die interaktive PDF bevorzugt wird. Soll die Textdatei für die Angabe der Daten verwendet werden, so muss die PDF umbenannt oder gelöscht werden. In diesem Fall würde das Programm auf die Textdatei zugreifen.

6.3. Einbinden der PDF in das Jenkins-Skript

Für eine Unterscheidung zwischen den beiden verschiedenen Arten der Konfigurationsdatei muss das Python-Skript, welches in Jenkins eingebunden wurde, angepasst werden. Dazu wird zunächst eine Funktion geschrieben, welche die vorhandene Konfigurationsdatei ermittelt. Diese ist in Abbildung 6.2 zu sehen.

```
def getConfigFile(campaign):
    if os.path.exists(campaignPath + '/data.pdf'):
        configFile = '/data.pdf'
    elif os.path.exists(campaignPath + '/data.txt'):
        configFile = '/data.txt'
    return configFile
```

Listing 6.2: Funktion zum Unterscheiden der Konfigurationsdateien

In dieser Funktion wird die PDF-Datei vorrangig behandelt. Es wird zunächst überprüft, ob eine PDF-Datei mit dem Namen "data.pdf" in dem Ordner der betrachteten Kampagne vorhanden ist. Ist dies der Fall, so wird der Name dieser Datei zurückgegeben. Andernfalls wird überprüft, ob eine Textdatei mit dem Namen "data.txt" in dem Ordner vorhanden ist. In diesem Fall wird diese in dem weiteren Verlauf verwendet.
Des Weiteren wird das Dictionary mit den benötigten Parametern auf die Parameter verkürzt, welche für die Verbindung mit der Datenbank benötigt werden. Diese Parameter werden zudem in einer Liste zusammengefasst, welche für das Einlesen der

Daten verwendet wird. Für das Einlesen wird die Funktion `readInData(campaignPath)` angepasst. Hier wird in dem ersten Schritt die Funktion aus dem Codeausschnitt 6.2 aufgerufen. Ist der Rückgabewert eine Textdatei, so wird der Code aus der vorherigen Version übernommen. Der Unterschied liegt darin, dass vor dem Schreiben der Parameter in ein Dictionary überprüft wird, ob dieser Parameter in der Liste mit den benötigten Daten vorhanden ist. Bei einer PDF-Datei wird das Einlesen ähnlich zu dem Einlesen der Konfigurationsdatei in dem Berichtgenerator durchgeführt. Dabei wird ebenfalls überprüft, wenn ein Wert betrachtet wird, ob dieser in der Liste mit den wichtigen Parametern enthalten ist. Ist dies der Fall, wird der Wert in das Dictionary eingetragen.
Weitere Anpassungen sind lediglich an den Stellen zu finden, an denen auf die Konfigurationsdatei zugegriffen werden muss. Hier wird jedes Mal auf die neue Funktion zugegriffen, welche den Namen der Datei zurück gibt. Dies ist notwendig, da die Kampagnen in verschiedenen Schleifen durchgegangen werden.

7. Zusammenfassung

Im Folgenden wird zunächst die Anwendung der Automatisierung beschrieben. Hierbei wird auf den Aufbau der virtuellen Maschine eingegangen und es wird erläutert, wie der Benutzer den Bericht einer neuen Kampagne erstellen kann. Anschließend wird der Ablauf beschrieben, was bei der Automatisierung geschieht. Hierzu wird das Ablaufdiagramm in Abbildung 7.3 erläutert.

7.1. Anwenden der Automatisierung

Die Automatisierung wird mit Hilfe von Jenkins auf der virtuellen Maschine durchgeführt. Die Ordnerstruktur der virtuellen Maschine ist in Abbildung 7.1 zu sehen. Hier ist zu sehen, dass zwei Hauptordner auf der virtuellen Maschine für die Automatisierung relevant sind. Dies ist zum einen das eingebundene Projektlaufwerk und zum anderen das Arbeitsverzeichnis von Jenkins. An dem Arbeitsverzeichnis von Jenkins muss der Benutzer nichts anpassen. Hier liegt das Python-Skript, welches in regelmäßigen Intervallen von Jenkins ausgeführt wird. Zudem liegen hier die Ordner der Kampagnen, welche von der Automatisierung betrachtet werden. Das Kopieren der Ordner findet durch das Python-Skript statt.

Das Projektlaufwerk ist für den Benutzer relevant. Hier liegt eine Vorlage für den Berichtgenerator. Diese Vorlage kann für eine neue Kampagne kopiert und angepasst werden. In der Abbildung 7.1 sind beispielsweise die Kampagnen "2022_ DHBW_ Precampaign", "2022_ DHBW_ Hauptcampaign" und "2022_ DHBW_ Delta-Campaign" zu sehen. Hierbei handelt es sich um die Ordner der entsprechenden

Kampagnen, welche ein Duplikat der Vorlage des Testberichts beinhalten. Die Konfigurationsdatei innerhalb des Ordners muss an die Daten des Kunden angepasst werden. Die Ordner der einzelnen Berichte können auch in weiteren Unterordnern auf dem Projektlaufwerk untergebracht sein.

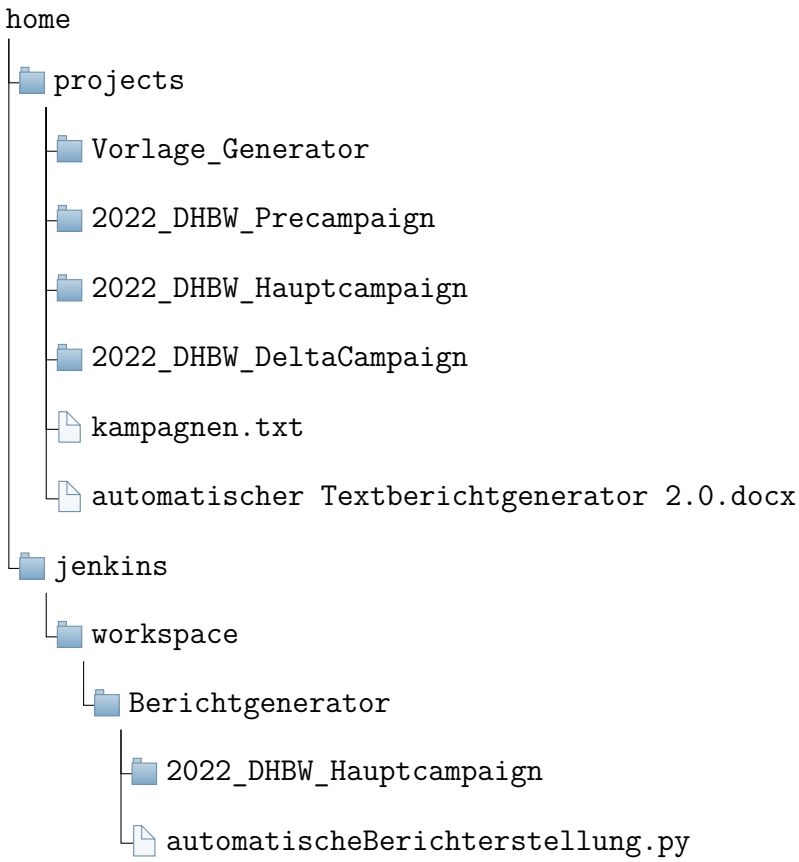


Abbildung 7.1.: Ordnerstruktur der virtuellen Maschine mit eingebundenem Laufwerk

Anschließend kann die Kampagne in die Automatisierung eingebunden werden. Hierzu muss der Pfad zu dem Ordner des jeweiligen Berichts in der Datei "kampagnen.txt" eingebunden werden. Das bedeutet, dass der Pfad in eine neue Zeile der Datei geschrieben wird. Ein Auszug dieser Datei ist in der Abbildung 7.2 zu sehen.

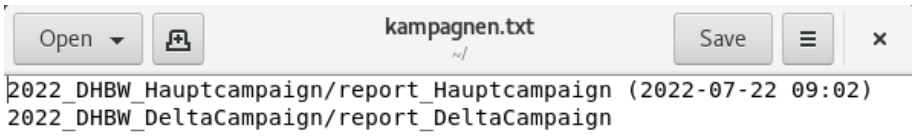


Abbildung 7.2.: Auszug aus der Textdatei mit den Kampagnen

Hier ist in der ersten Zeile die Hauptkampagne zu sehen. Da hinter dem Pfad der Kampagne bereits ein Zeitstempel angegeben ist, wurde dieser Bericht bereits mindestens einmal automatisch erstellt. Die nächste Erstellung wird durchgeführt, wenn der Zeitstempel der Konfigurationsdatei oder der Datenbank ein aktuellerer ist oder wenn neue lokale Daten vorliegen. In diesem Fall wird der Zeitstempel hier vor der Erstellung angepasst. Die angegebene Delta-Kampagne wurde neu zu der Datei hinzugefügt. Aus diesem Grund ist sie in der Abbildung 7.1 ausschließlich auf dem Projektlaufwerk zu sehen. Bei dem nächsten Durchlauf des Python-Skripts wird der Ordner des Berichts dieser Kampagne ebenfalls in das Arbeitsverzeichnis von Jenkins kopiert und es wird ein Zeitstempel in der Textdatei eingefügt. Anschließend kann der Bericht dieser Kampagne erzeugt werden. Zu finden ist das Ergebnis am Ende in dem richtigen Ordner auf dem Projektlaufwerk.
Die erste Kampagne auf dem Projektlaufwerk ist lokal auch nicht zu sehen. Hier wird sichtbar, dass ein Ordner einer Kampagne lokal gelöscht wird, sobald er in der Textdatei nicht mehr erscheint. Dies ist hier der Fall, da in der Datei nur die anderen beiden Kampagnen aufgezählt werden.
Auf dem Laufwerk befindet sich zusätzlich eine Word-Datei. Dabei handelt es sich um die Dokumentation zu dem Berichtgenerator, welche dem Benutzer nochmals die Funktionsweise darlegt.

7.2. Ablauf der Berichterstellung

In Abbildung 7.3 ist der Ablauf der Berichterstellung zu sehen. Dabei wird dargelegt, welche Überprüfungen und Entscheidungen von dem Python-Skript getroffen werden, welches in Jenkins eingebunden ist. Der Ablauf ist hierbei für jede Kampagne einzeln zu betrachten. In dem Jenkins-Skript werden die Kampagnen durchgegangen und

für jede wird dieser Ablauf durchgeführt. Abfragen für das Abfangen von Fehlern werden hierbei nicht betrachtet.

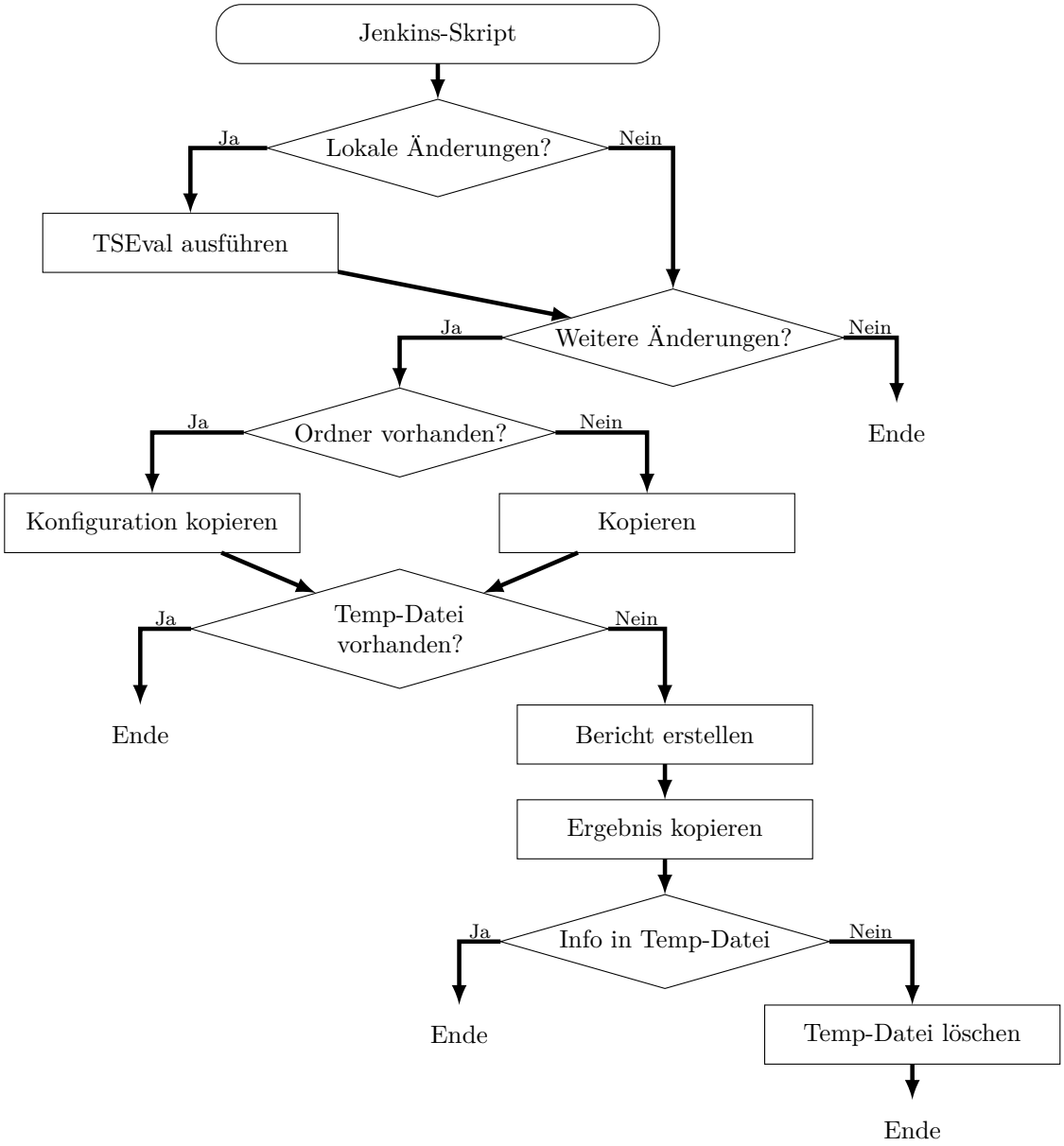


Abbildung 7.3.: Ablaufdiagramm für die automatische Berichterstellung

In dem ersten Schritt wird überprüft, ob lokale Änderungen an den Sequenzen vorgenommen wurden. Ist dies der Fall, wird TSEval für diese Daten ausgeführt.

Unabhängig von der gegebenen Antwort wird in dem nächsten Schritt geprüft, ob weitere bzw. andere Änderungen an den Daten vorgenommen wurden. Wenn ja, so wird zusätzlich überprüft, ob der Ordner der Kampagne bereits lokal vorhanden ist. Entsprechend muss entweder der ganze Ordner oder nur die Konfigurationsdatei kopiert werden. In dem nächsten Schritt wird die Kampagne auf die temporäre Datei überprüft. Ist diese auf dem Projektlaufwerk vorhanden, so wird der Bericht bereits erstellt oder bei der letzten Erstellung kam es zu einem Fehler. In diesem Fall wird die Berichterstellung nicht nochmal neu gestartet. Andernfalls wird der Bericht erstellt und die Ergebnisse anschließend auf das Projektlaufwerk kopiert, wenn neue Ergebnisse entstanden sind. In dem letzten Schritt wird die temporäre Datei auf dem Projektlaufwerk eingelesen. Sind hier Informationen vorhanden, bleibt die Datei bestehen und die Berichterstellung ist für diese Kampagne beendet. Andernfalls kann die Datei gelöscht werden, sodass bei den nächsten Änderungen die Berichterstellung problemlos neu gestartet werden kann.

8. Abschließende Betrachtungen

In diesem Kapitel werden abschließende Betrachtungen angeführt, welche die Arbeit zusammenfassen, die Performance aufgreifen und einen Ausblick geben. Hierbei wird auf Verbesserungen und Erweiterungen eingegangen, welche nach Abschluss dieser Arbeit umgesetzt werden können.

8.1. Fazit

Im Rahmen dieser Arbeit wurde die "Automatisierung der Berichterstellung für Konformitätstests von Zugsicherungskomponenten" betrachtet. Hierbei ging es hauptsächlich um die Vereinfachung und Benutzerfreundlichkeit des Berichtgenerators. Umgesetzt und angepasst wurde in diesem Rahmen das Konzept aus einer vorherigen Arbeit. Zudem wurde das Konzept detaillierter ausgearbeitet für die Erstellung der Konfigurationsdateien und für die Umsetzung der Automatisierung an sich. Dabei wurden die meisten Fehler und Ausnahmen abgefangen.
Bei der Umsetzung wurde zunächst eine Konfigurationsdatei angefertigt, um das Anpassen des Berichtes an die entsprechende Kampagne zu vereinfachen. Diese wurde in dem ersten Schritt in Form einer Textdatei angefertigt, um die Automatisierung umsetzen zu können. In dem finalen Berichtgenerator kann der Benutzer zwischen der Konfigurationsdatei in Form einer Textdatei und in Form einer interaktiven PDF wählen. Dabei wird die interaktive PDF bevorzugt, wenn beide Dateien in dem Ordner vorhanden sind. Möchte der Nutzer die Textdatei verwendet, so muss die PDF zuvor gelöscht werden. Mit Hilfe der Konfigurationsdatei konnte anschließend die Automatisierung umgesetzt werden. Diese läuft über Jenkins. Hier wird alle 30

Minuten ein Skript auf der virtuellen Maschine ausgeführt, welches in einer Textdatei die Pfade der Kampagnen einliest, von welchen die Berichte erstellt werden sollen. Anschließend werden die Konfigurationsdatei, die Datenbank und die lokalen Daten dieser Kampagnen auf Änderungen überprüft und der Testbericht wird neu erstellt, wenn Änderungen vorliegen. Hierzu wird der Ordner der Kampagne von dem Projektlaufwerk auf die virtuelle Maschine kopiert. Auf dieser kann der Bericht lokal erstellt werden, sodass keine anderen Ressourcen belastet werden.
Der Berichtgenerator und das Automatisierungsskript wurden so angepasst, dass bestimmte Fehler abgefangen werden. Hierzu zählt eine nicht erfolgreiche Berichterstellung und Änderungen an Kampagnen in kurzen Zeitabständen. Um einen Bericht nicht zeitgleich zweimal zu erstellen, wurde eine temporäre Datei eingebunden. Diese existiert so lange in dem Ordner der Kampagne, bis der Bericht fertig erstellt wurde. Eine Ausnahme bildet eine fehlerhafte Berichterstellung. Hierbei wird die Datei verwendet, um die Fehler in dieser zu notieren. Dabei wird die Datei am Ende der Erstellung nicht gelöscht und der Nutzer kann die Fehlermeldungen einsehen. Weitere Erweiterungen sind zum einen das Threading, sodass alle Ressourcen der virtuellen Maschine ausgenutzt werden, wenn mehrere Berichte zeitgleich erstellt werden können. Zum anderen werden alte Kampagnen lokal gelöscht, sobald diese nicht mehr betrachtet werden.

8.2. Performanzbetrachtungen

Abschließend zu der Zusammenfassung wird die Performanz der Berichterstellung betrachtet. Hierzu wurden Anwender befragt, welche aktiv in diesem Bereich tätig sind. Für die bisherige Berichterstellung wurde nach dem Bearbeiten der Sequenzen eine zusätzliche Woche eingeplant. Dabei wird hier das erste Mal ein Bericht zu der Kampagne erstellt. In dem ersten Schritt müssen die Daten des Herstellers gesammelt werden. Anschließend werden die Daten der Sequenzen in die Datenbank eingebunden, was von den Anwendern auf eine Zeitspanne von 12 bis 24 Stunden eingeschätzt wurde. Danach kann die Berichterstellung gestartet werden. Die Dauer des Skripts wurde mit zehn bis zwölf Stunden beschrieben, je nach Kampagne können die Zeiten

variieren. Dabei wird zum einen der Testbericht selbst erstellt, aber auch einzelne PDF-Dateien zu allen Sequenzen, welche die Ergebnisse dieser beinhalten. Diese *trace*-Dateien beanspruchen bei der Ausführung des Skripts die meiste Zeit, während die Erstellung des Berichts selbst nur wenige Sekunden in Anspruch nimmt.

Werden nach der Erstellung des Testberichts nochmals Änderungen an einzelnen Sequenzen vorgenommen, so müssen die Daten von diesen neu in die Datenbank geladen werden. Hierzu wird das Programm TSEval mit den Angaben für die Verbindung mit der Datenbank für die entsprechenden Sequenzen gestartet. Es existiert ein Automatisierungsskript, welches alle Sequenzen iteriert und für jede TSEval ausführt. Dabei ist der Nachteil, dass nochmal alle Sequenzen neu in die Datenbank geladen werden, was erneut bis zu einem Tag beansprucht. Sollen nicht alle Daten erneut in die Datenbank geladen werden, kann für die veränderten Sequenzen händisch TSEval gestartet werden. Dabei hat der Anwender die Arbeit, da alle Sequenzen nacheinander betrachtet werden müssen.

Durch die Automatisierung kann bereits in vielen Aspekten Zeit eingespart werden. Die Daten des Herstellers müssen weiterhin zu Beginn gesammelt werden, was jetzt jedoch direkt zu Beginn der Kampagne durchgeführt werden soll. So kann der Bericht regelmäßig erstellt und hierdurch Zeit am Ende eingespart werden. Das Laden der Daten in die Datenbank wird durch die Automatisierung direkt durchgeführt. Sobald neue Sequenzen ausgewertet oder Änderungen vorgenommen werden, wird TSEval für die entsprechende Sequenz mit einer Verbindung zu der Datenbank ausgeführt. Durch diesen Vorgang werden zum Ende der Kampagne bis zu 24 Stunden gespart. Die Berichtgenerierung dauert weiterhin mehrere Stunden, da bei der Erstellung der einzelnen *trace*-Dateien noch keine Verbesserung implementiert wurde. Hierbei ist durch das direkte Laden der Daten in die Datenbank bereits ein Anfang umgesetzt. Im Weiteren kann betrachtet werden, dass nur von diesen Sequenzen eine neue Datei generiert wird. Durch diese Anpassung können weitere 12 Stunden eingespart werden.

Ein großer Vorteil der Automatisierung liegt in der regelmäßigen Erstellung der Testberichte von Beginn der Kampagne an. Hierdurch kann der Hersteller der OBU regelmäßig Zwischenberichte erhalten. Diese dienen der Transparenz zwischen dem Auftraggeber und dem DLR. Weiter kann der Hersteller so den Fortschritt bei

dem Testen offen einsehen, ohne dass während der Kampagne weitere Zeit in die Berichterstellung investiert werden muss.

Schlussendlich kann am Ende einer Kampagne weniger als eine Woche Zeit eingeplant werden, da mindestens ein Tag eingespart werden kann. Auch bei weiteren Änderungen nach der Berichterstellung müssen nicht nochmals bis zu 2 Tage eingeplant werden, um den Bericht zu aktualisieren. Hier wird ebenfalls bis zu einem Tag eingespart. Durch die bereits erläuterte Erweiterung kann die Dauer der Testberichterstellung am Ende einer Kampagne um bis zu 36 Stunden reduziert werden. Bei weiteren Änderungen kann dann nur die veränderte Sequenz erneut in die Datenbank geladen werden und auch nur zu dieser muss eine neue *trace*-Datei erstellt werden.

8.3. Ausblick

Anschließend an diese Arbeit können weitere Anpassungen an dem Berichtgenerator vorgenommen werden. Hierbei geht es um eine Laufzeitverbesserung, aber auch um weitere Anpassungen zur Benutzerfreundlichkeit. Bisher konnten diese Erweiterungen aufgrund von fehlender Zeit nicht umgesetzt werden.

Für eine verbesserte Laufzeit kann das Laden der Daten aus der Datenbank angepasst werden. Hierbei ist es nicht notwendig, Daten neu herunterzuladen, welche seit dem letzten Download nicht verändert wurden. Dabei können die Zeit der letzten Änderung in der Datenbank und die Zeit, zu welcher die Daten heruntergeladen wurden, verglichen werden. Ist das Änderungsdatum aktueller, so müssen die Daten aktualisiert werden. Hierbei geht es um die Erstellung der *trace*-Dateien, wobei viel Zeit investiert werden muss, da bisher immer alle neu erstellt werden müssen. Durch diese Erweiterung können nur die *trace*-Dateien erstellt werden, bei welchen Änderungen vorliegen.

Weitere Anpassungen können an der interaktiven PDF vorgenommen werden. Zum einen kann ein Makro eingebunden werden, welches auf die Abfrage der Delta-Kampagne reagiert. Hier kann dafür gesorgt werden, dass die Freitexte nur dann bearbeitet werden können, wenn es sich um eine Delta-Kampagne handelt. Zum

anderen kann ein weiteres Makro implementiert werden, durch welches die Daten in der Konfigurationsdatei direkt überprüft werden. So bekommt der Benutzer direkt ein Feedback, sobald ein Fehler vorliegt. Wichtig ist dabei, dass alle Felder der Datei ausgefüllt sind. Zudem ist besonders bei den Daten für die Verbindung mit der Datenbank die Korrektheit der Einträge wichtig. Auch hier kann zumindest überprüft werden, ob das Format der angegebenen Daten das Richtige ist.
Auch die Konfigurationsdatei in Form einer Textdatei kann bei dem Einlesen in das Skript überprüft werden. Wird bei diesem Dokument die Formatierung nicht eingehalten, können Fehler entstehen. Diese bekommt der Benutzer in dem schlimmsten Fall nicht mitgeteilt. Daher kann hier ebenfalls eine Kontrolle eingebunden werden, durch welche die Fehler notiert werden können.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich in der Zeit meiner Bachelorarbeit unterstützt haben.

Zuerst möchte ich mich bei meinem Gruppenleiter Dipl.-Inf. Oliver Röwer bedanken, der mir bei der Findung der Aufgabenstellung geholfen hat und in Bezug auf inhaltliche Fragen immer für mich da war. Zudem möchte ich mich für die Korrektur meiner Bachelorarbeit in Blick auf die Richtigkeit des theoretischen Hintergrundes bedanken.

Ich bedanke mich bei meinem betrieblichen Betreuer Dipl.-Ing. Lennart Asbach für die organisatorische Unterstützung und für die konstruktive Kritik zur Verbesserung meiner Arbeit.

Zudem möchte ich mich bei meinem Betreuer der dualen Hochschule Prof. Dr. Nathan Sudermann-Merx für die Hilfe bei der Strukturierung der Bachelorarbeit und für die formalen Korrekturen bedanken.

Ebenfalls möchte ich mich bei meinen Kommilitonen bedanken, die mir mit viel Geduld zur Seite gestanden haben und mich bis zum Schluss in vielen Situationen unterstützt haben.
Auch für das viele Korrekturlesen meiner Arbeit möchte ich mich herzlich bei meinen Kommilitonen und Freunden bedanken.

Literatur

[1] Deutsche Akkreditierungsstelle GmbH (DAkkS). „Akkreditierung“. In: (2020).

[2] DB Netz AG. „Glossar“. In: *Europäisches Zugbeeinflussungssystem (ETCS)* (2014).

[3] Deutsche Bahn AG. *ETCS: Das Europäische Zugsicherungssystem*. 2020. URL: https://inside.bahn.de/etcs-europaeisches-zugsicherheitssystem/ (besucht am 23. 05. 2022).

[4] Deutsche Bahn AG. *Rechtliche Grundlagen*. 2019. URL: https://fahrweg.dbnetze.com/fahrweg-de/kunden/nutzungsbedingungen/etcs/Grundlagen-ETCS/Rechtliche-Grundlagen-7577576?contentId=7576626 (besucht am 23. 05. 2022).

[5] Deutsche Akkreditierungsstelle. *Was ist Akkreditierung?* 2022. URL: https://www.dakks.de/de/was-ist-akkreditierung.html (besucht am 23. 08. 2022).

[6] Lars Brune et al. *ETCS an Landesgrenzen: Interoperabilität und Ausrüstungsvarianten*. 2021. URL: https://fahrweg.dbnetze.com/resource/blob/6355844/b17da390c35216b9c6fc8728d0a5be5a/ETCS-an-Landesgrenzen-Brune_Kahnert_Kalkreiber_Lens--data.pdf (besucht am 22. 08. 2022).

[7] Jim Anderson. *An Intro to threading in Python*. 2019. URL: https://realpython.com/intro-to-python-threading/ (besucht am 21. 07. 2022).

[8] An Agency of the European Union. *Set of specifications 3 (ETCS B3 R2 GSM-R B1) | ERA*. 2022. URL: https://www.era.europa.eu/content/set-specifications-3-etcs-b3-r2-gsm-r-b1 (besucht am 23. 08. 2022).

[9] Wikipedia Foundation. *Polling (Informatik)*. 2006. URL: https://de.wikipedia.org/wiki/Polling_(Informatik) (besucht am 16. 06. 2022).

[10] Computer Futures. *Wie gelingt die Automatisierung von Geschäftsprozessen?* 2021. URL: https://www.computerfutures.com/de-de/blog/2021/03/wie-gelingt-die-automatisierung-von-geschaeftsprozessen/ (besucht am 03. 08. 2022).

[11] GeeksforGeeks. *How to compare two text files in python?* 2021. URL: https://www.geeksforgeeks.org/how-to-compare-two-text-files-in-python/ (besucht am 07. 06. 2022).

[12] Guru99. *Python File Handling: How to Create Text File, Read, Write, Open.* 2022. URL: `https://www.guru99.com/reading-and-writing-files-in-python.html` (besucht am 19. 07. 2022).

[13] Justin. *Interactive PDF is dead - here's what you can create from InDesign that's even better.* 2018. URL: `https://ajarproductions.com/blog/2018/03/26/interactive-pdf-is-dead-heres-what-you-can-create-from-indesign-thats-even-better/` (besucht am 04. 07. 2022).

[14] Michelle Knop. „Dokumentation: Generator zur automatischen RailsiTe-Testberichterstellung 2.0". In: (2022).

[15] Michelle Knop. „Praxisbericht: Entwicklung eines Konzepts zur Automatisierung der Testberichterstellung". DHBW Mannheim, 2022.

[16] Michelle Knop. „Praxisbericht: Modularisierung der Evaluierungsfunktionalitäten des Verifikations- und Validierungswerkzeugs TSEval und Erweiterung um eine Datenbankfunktion". DHBW Mannheim, 2021.

[17] Die Europäische Kommission. „Durchführungsverordnung (EU) 2019/776 der Kommission vom 16. Mai 2019 zur Änderung der Verordnungen (EU) Nr. 321/2013, (EU) Nr. 1299/2014, (EU) Nr. 1301/2014, (EU) Nr. 1302/2014, (EU) Nr. 1303/2014 und (EU) 2016/919 der Kommission sowie des Durchführungsbeschlusses 2011/665/EU der Kommission im Hinblick auf die Angleichung an die Richtlinie (EU) 2016/797 des Europäischen Parlaments und des Rates und Umsetzung der in dem Delegierten Beschluss (EU) 2017/1474 der Kommission festgelegten spezifischen Ziele". In: (2019).

[18] Oliver Röwer Lennart Asbach. „Allgemeine Vorlage eines Testberichts". In: (2022).

[19] Deutsches Zentrum für Luft- und Raumfahrt. *DLR Events | RailSiTe®.* 2020. URL: `https://event.dlr.de/zug-zur-digitalisierung/zzd-exponat09/` (besucht am 24. 05. 2022).

[20] Deutsches Zentrum für Luft- und Raumfahrt e.V. *DLR - Institut für Verkehrssystemtechnik - RailSiTe®: Eisenbahntechnisches Simulations- und Testlabor.* 2022. URL: `https://www.dlr.de/ts/desktopdefault.aspx/tabid-11368/19985_read-3254/` (besucht am 23. 05. 2022).

[21] Sanjay Mohapatra. *Business Process Automation.* New Dehli, 2009.

[22] Martin Schiele. „Generator zur automatischen RailsiTe-Testberichterstellung". In: (2021).

[23] Sebastian Stein. *Kommandos definieren | Eine kleine LaTeX (Tex) Einführung.* o. J. URL: `http://latex.hpfsc.de/content/latex_tipps_und_tricks/eigene_kommandos/` (besucht am 09. 06. 2022).

[24] TechTarget. *Was ist Business Process Automation (BPA)? - Definition von WhatIs.com*. 2022. URL: https://www.computerweekly.com/de/definition/Business-Process-Automation-BPA (besucht am 23. 08. 2022).

[25] o. V. *.whl Dateierweiterung*. o. J. URL: https://datei.wiki/extension/whl (besucht am 05. 07. 2022).

[26] o. V. *1 latex unterstrich*. o. J. URL: https://www.namsu.de/Extra/befehle/Unterstrich.html (besucht am 14. 07. 2022).

[27] o. V. *Extracting Filled Out Fielöds From a PDF*. 2018. URL: https://www.reddit.com/r/learnpython/comments/9svk1o/extracting_filled_out_fields_from_a_pdf/ (besucht am 11. 07. 2022).

[28] o. V. *How To Create Fillable PDF Forms With LibreOffice Writer*. o. J. URL: https://www.linuxuprising.com/2019/02/how-to-create-fillable-pdf-forms-with.html (besucht am 04. 07. 2022).

[29] o. V. *How to get the latest file in a folder?* 2016. URL: https://stackoverflow.com/questions/39327032/how-to-get-the-latest-file-in-a-folder (besucht am 28. 06. 2022).

[30] o. V. *installing python packages without internet and using source code as .tar.gz and .whl*. 2016. URL: https://stackoverflow.com/questions/36725843/installing-python-packages-without-internet-and-using-source-code-as-tar-gz-and (besucht am 05. 07. 2022).

[31] o. V. *Run pdflatex quietly [closed]*. 2009. URL: https://stackoverflow.com/questions/1037927/run-pdflatex-quietly (besucht am 21. 07. 2022).

[32] o. V. *What is the difference between interaction=nonstopmodeänd halt-on-error"?* 2015. URL: https://tex.stackexchange.com/questions/258814/what-is-the-difference-between-interaction-nonstopmode-and-halt-on-error (besucht am 28. 06. 2022).

[33] DLR - Institut für Verkehrssystemtechnik. „RAILSITE-HANDBUCH“. In: (2022).

[34] Ryan Wells. *How to Read PDF Files with Python using PyPDF2*. 2021. URL: https://wellsr.com/python/read-pdf-files-with-python-using-pypdf2/ (besucht am 07. 06. 2022).

Anhang A.

Bilder

A.1. Konfigurationsdatei (Textdokument)

```
#####parameterCreation:
CustomerFiles: DHBW

reusePDFtraces: no
newPDFtraces: yes
compilePDFfile: yes
deltaCampagne: yes

dbname:
dbhost:
dbport:
dbuser:
dbpasswd:


#####parameterReport:
Customer: Duale Hochschule Baden-Würtemberg
CustomerShort: DHBW
CustAddressA: Duale Hochschule Baden-Würtemberg
CustAddressB: Coblitzallee 1-9
CustAddressC: 68163 Mannheim
CustAddressD: Deutschland
Unit: Test-Unit
Release: 1.5-4.0
UnitVersionA: Product Version Information: -
UnitVersionB: Product Number: -
SerialNo: 123456789
DeliveryDate: January 2022
DateString: 13 July 2022
DateSignature: 13/07/2022

Version: 1.0
RepID: DHBW-123456-789
PL: Max Mustermann
HOD: Maxime Mustermann
Authors: Michelle Knop
Period: July 2022
RailsiteConfig: DHBW - 98765 - 43210
SubsetVersion: v1.2.3
SubsetTestSpecVersion: v1.2.3
SubsetJRUVersion: v1.2.3
SubsetDMIVersion: v1.2.3
TotalNumSeq: 745
```

Abbildung A.1.: Auszug aus der Konfigurationsdatei in Form einer Textdatei (Teil 1)

```
#####modificationHistory:
v1.0, 11/02/2022: Initial Version;


#####chapter1-disclaimerAndExtensions:
This report does not represent a full conformity test report.
This test report is based on a partial test of Subset-076-6-3
\SubsetTestSpecVersion~\cite{Subset76Seq}.
The part that is executed has been selected by the customer.
The partial test has been performed under the same constraints that a full conformity test
requires.


#####chapter2-introduction:
The test sequences which are executed have been selected by the customer.
This partial test does not represent a full Subset-076-6-3
\SubsetTestSpecVersion~\cite{Subset76Seq} conformity test.


#####chapter3-1-unitUnderTest:
The customer was solely responsible for the selection, provision and configuration of the test
candidate, i.e. to provide a test sample. The on-board unit was originally delivered by the
customer to the test laboratory on {\DeliveryDate}. The actually used configurations, supported
interfaces and software details of the on-board unit are listed in the integration
report~\cite{IntegrationReportALS2022_P8cMR9}.


#####chapter3-2-testScope:
The test sequences have been selected by the customer.
```

Abbildung A.2.: Auszug aus der Konfigurationsdatei in Form einer Textdatei (Teil 2)

A.2. Konfigurationsdatei (interaktive PDF)

Konfigurationsdatei für die Berichterstellung

In dieser Datei werden die Daten abgefragt, welche für die Berichterstellung benötigt werden. Dazu gibt es zuerst Daten, welche für das Erstellen des Berichts notwendig sind. Anschließend folgen Daten, welche in den Bericht eingebunden werden. Schlussendlich werden Kapitelinhalte angegeben, welche für Delta-Testkampagnen in den Bericht eingebunden werden müssen.

Parameter für die Erstellung

CustomerFiles: DHBW

(Verwendet für die Benennung von automatisch generierten Dateien, welche Kundendaten enthalten)

Fragen zu der Berichterstellung:

1. "x" alte PDF-trace(s) wurden gefunden.
   Sollen diese wiederverwertet werden? ◉ Ja ○ Nein
   (ACHTUNG: Die alten PDF-traces passen möglicherweise nicht zu den Daten in der Datenbank!)
2. Sollen neue PDF-traces erstellt werden? ○ Ja ◉ Nein
3. Soll die finale PDF-Datei generiert werden? ◉ Ja ○ Nein

Angaben für die Verbindung mit der Datenbank:

Datenbankname:
Host:
Port:
Benutzer:
Passwort:

Abbildung A.3.: Auszug aus der Konfigurationsdatei in Form einer interaktiven PDF (Teil 1)

Parameter zum Eindinden in den Bericht

Name des Kunden:	Duale Hochschule Baden-Würtemberg
Kurzform für den Namen des Kunden:	DHBW
Adresse des Kunden:	Duale Hochschule Baden-Würtemberg
	Coblitzallee 1-9
	68163 Mannheim
	Deutschland
Unit:	Test-Unit
(Name der zu testenden On-Board Unit)	
Release:	1.5-4.0
(???)	
Unit-Version A:	Product Version Information: -
Unit-Version B:	Product Number: -
Serien Nummer:	123456789
Anlieferungsdatum:	January 2022
Datum:	13 July 2022
Datum der Signierung:	13/07/2022
Version:	1.0
Berichts-ID:	DHBW-123456-789
Technical Review (Projektleiter):	Max Mustermann
(Prüfung auf technische Richtigkeit)	
Approval (Stellvertretender Laborleiter):	Maxime Mustermann
(Genehmigung)	
Autoren:	Michelle Knop
Zeitraum:	July 2022
RailsiteConfig:	DHBW - 98765 - 43210
(???)	
Subset-Version:	v1.2.3
Subset Testspezifikation-Version:	v1.2.3
Subset JRU-Version:	v1.2.3
Subset DMI-Version:	v1.2.3
Gesamtzahl der Sequenzen:	913

Abbildung A.4.: Auszug aus der Konfigurationsdatei in Form einer interaktiven PDF (Teil 2)

Freitexte in dem Bericht

Dem Testbericht sind Textpassagen enthalten, welche ggf. Bearbeitet werden müssen. Dabei gibt es zunächst eine Midifikations-Historie, welche im Folgenden angegeben und erweitert werden kann. Außerdem gibt es zusätzliche Texte, welche eingebunden werden müssen, wenn es in dem Bericht um eine Delta Kampagne handelt. In diesem Fall muss angegeben werden, dass es sich nicht um eine vollständige Kampagne handelt und der Kunde die Testsequenzen ausgewählt hat. Sobald hierbei angegeben wird, dass es sich um eine Delta-Kampagne handelt, können die Freitexte bearbeitet werden.

ModifikationsHistorie:

(Soll eine neue Version der Historie hinzugefügt werden, muss diese in einer neuen Zeile angefügt werden. Getrennt wird die Versionsnummer von der Beschreibung mit einem Doppelpunkt)

v1.0, 11/02/2022: Initial Version

Handelt es sich um eine Delta-Kampagne mit ausgewählten Testsequenzen? ◉ Ja ○ Nein

Kapitel 1: „Disclaimer and Extensions“

This report does not represent a full conformity test report.
This test report is based on a partial test of Subset-076-6-3 \SubsetTestSpecVersion~\cite{Subset76Seq}.
The part that is executed has been selected by the customer.
The partial test has been performed under the same constraints that a full conformity test requires.

Kapitel 2: „Introduction“

The test sequences which are executed have been selected by the customer.
This partial test does not represent a full Subset-076-6-3 \SubsetTestSpecVersion~\cite{Subset76Seq} conformity test.

Abbildung A.5.: Auszug aus der Konfigurationsdatei in Form einer interaktiven PDF (Teil 3)

Kapitel 3.1: „Unit Under Test“

The customer was solely responsible for the selection, provision and configuration of the test candidate, i.e. to provide a test sample. The on-board unit was originally delivered by the customer to the test laboratory on {\DeliveryDate}. The actually used configurations, supported interfaces and software details of the on-board unit are listed in the integration report~\cite{IntegrationReportALS2022_P8cMR9}.

Kapitel 3.2: „Test Scope“

The test sequences have been selected by the customer.

Abbildung A.6.: Auszug aus der Konfigurationsdatei in Form einer interaktiven PDF (Teil 4)

Anhang B.

Codeausschnitte

```
def readConf():
    # read file in complete text
    with open(parameterFile) as txt:
        text=txt.read().rstrip('\n')
    # split the text at the seperator '#####'
    dataTemp=text.split('#####')

    # save the entrys of the text in a dictionary
    count = 0
    # iterate the sections
    for line in dataTemp:
        # split the section in header and body
        dataTemp[count]=line.split(': ',1)
        # if there is any body:
        if len(dataTemp[count]) == 2:
            header=dataTemp[count][0]
            body=dataTemp[count][1]

            # sections for the free-text
            if header in sections:
                # the first new line has to be cut off
                dictionary[header]=body.split('\n',1)[1]
            # modify the modificationhistory
            elif header == 'modificationHistory':
                # the first new line has to be cut off
                modificationHistory=body.split('\n',1)[1]
                # cut off the empty lines at the end after the
                    semicolon
                modificationHistory=modificationHistory.split(
                    ';')[0]
                # modify the data for table entrys
                modificationHistory=modificationHistory.
                    replace(':',' &')
                modificationHistory=modificationHistory.
                    replace('\n','\\\\')
                dictionary[header]=modificationHistory
            # write the parameter to the dictionary
            else:
                # split the section line by line in separate
                    parameters
```

```
                    tempSection=body.split('\n')
                    count2 = 0
                    for line in tempSection:
                        # split line in parameter and entry
                        tempSection[count2]=line.split(': ',1)
                        if len(tempSection[count2]) == 2:
                            parameter=tempSection[count2][0]
                            entry=tempSection[count2][1]
                            if not entry:
                                print('Parameter '+parameter+' not
                                    found.')
                            else:
                                if parameter in parameterReport:
                                    entry=entry.replace('_',r'\_')
                                dictionary[parameter]=entry
                            count2 += 1
                count += 1

        return dictionary
```

Listing B.1: Funktion zum Einlesen der Konfigurationsdatei in Form einer Textdatei