FERDINAND PORSCHE
FERN FH

Konzeption und prototypische Implementierung eines Frameworks für den Einsatz von Robotic Process Automation zur Testautomatisierung

Masterarbeit

Eingereicht von: **Jürgen Seisl, BA**

Matrikelnummer: 51807328

im Fachhochschul-Masterstudiengang Wirtschaftsinformatik

der Ferdinand Porsche FernFH GmbH

zur Erlangung des akademischen Grades

Master of Arts in Business

Betreuung und Beurteilung: Dipl.-Ing. Dr. Igor Miladinovic

Zweitgutachten: FH-Prof. Dipl.-Ing. Dr. techn. Dr. techn. Dr.-Ing. Gernot Kucera, MA

Leoben, Mai 2023

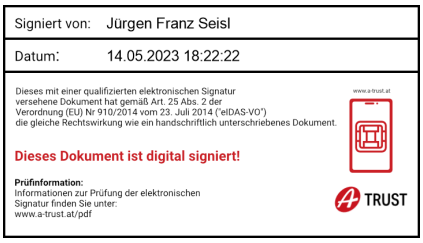
Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.
2. dass ich diese Masterarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe.
3. dass die vorliegende Fassung der Arbeit mit der eingereichten elektronischen Version in allen Teilen übereinstimmt.

Leoben, 14.5.2023 _____

Unterschrift



Kurzzusammenfassung: Konzeption und prototypische Implementierung eines Frameworks für den Einsatz von Robotic Process Automation zur Testautomatisierung

Für eine Testautomatisierung von Softwaretests stehen mehrere unterschiedliche Tools zur Verfügung. Oftmals gibt es aber im Unternehmen schon Erfahrung mit der Automatisierung von Prozessen mittels Robotic Process Automation. RPA kann zwar Prozesse automatisiert abarbeiten, ist aber kein Tool, welches für die Testautomatisierung erstellt wurde. Aus dieser Problemstellung heraus resultiert meine Forschungsfrage:

„Wie muss ein Framework für den Einsatz von Robotic Process Automation Tools zur Testautomatisierung bei Regressionstests im End-to-End-Bereich konzipiert sein, damit die Steuerung der einzelnen Test-Cases in Form von RPA-Prozessen sowie ein Test-Reporting der Ergebnisse ermöglicht wird?“

Für die Beantwortung wurde die Methode des Design Science nach Österle et al. gewählt. Dadurch war es möglich, in den vier iterativen Schritten, Analyse, Entwurf, Evaluierung und Diffusion, einen funktionalen Prototyp des Frameworks zu entwickeln. Die Evaluierung des Prototyps hat gezeigt, dass ein Framework so gestaltet sein muss, dass die Prozesse der Testautomatisierung auch mit dem Framework umgesetzt werden können, es für unterschiedliche RPA-Tools anwendbar ist, es möglich sein muss, die Testdurchläufe individuell anzupassen, es am Ende jedes Testdurchlaufes eine Übersicht über die Ergebnisse sowie Details über die Fortschritte bei den einzelnen Testfällen geben muss.

Schlagwörter:

Testautomatisierung, Regressionstest, Softwaretest, Robotic Process Automation, Softwareroboter

Abstract: Conception and prototypical implementation of a framework for the use of Robotic Process Automation for test automation

There are several different tools available for automating software tests. Often, however, the company already has experience with the automation of processes using robotic process automation. Although RPA can automate processes, but it is not a tool that was created for test automation. My research question stems from this problem statement:

"How must a framework for using Robotic Process Automation tools for test automation in end-to-end regression testing be designed to enable control of individual test cases in the form of RPA processes, as well as test reporting of results?"

For the answer, the Design Science method according to Österle et al. was chosen. This made it possible to develop a functional prototype of the framework in the four iterative steps of analysis, design, evaluation and diffusion. The evaluation of the prototype has shown that a framework must be designed in such a way that the processes of test automation can also be implemented with the framework, it must be applicable for different RPA tools, it must be possible to adapt the test runs individually, there must be an overview of the results at the end of each test run, as well as details about the progress of the individual test cases.

Keywords:

Test automation, regression testing, software testing, robotic process automation, software robots

Danksagung

Ich möchte mich bei allen Personen bedanken, die mich im Rahmen des gesamten Studiums und bei der Erstellung der Masterarbeit unterstützt haben.

Ein besonderer Dank gilt meiner Frau Nina sowie meinen beiden Kindern Lara-Franziska und Philipp, die mir sehr viel Geduld und Verständnis für mein Studium entgegengebracht haben.

Dipl.-Ing. Dr. Igor Miladinovic möchte ich für die Übernahme der Betreuung dieser Arbeit und für sein Feedback danken, welches sehr zur Verbesserung der Qualität dieser Arbeit beigetragen hat.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird die Motivation hinter diesem Thema für die Arbeit umrissen und die Problemstellung dazu ausgeführt. Im Anschluss wird eine Übersicht über die theoretischen Grundlagen zum Thema angeführt.

1.1 Motivation und Problemstellung:

MitarbeiterInnen sind eine der wichtigsten Ressourcen eines Unternehmens. Daher sollten MitarbeiterInnen nicht für einfache, monotone oder immer wiederkehrende Arbeit eingesetzt werden. Hinzukommt, dass Maschinenzeit wesentlich billiger als menschliche Arbeitszeit ist. Aus diesem und vielen weiteren Gründen setzen Unternehmen zunehmend auf Automatisierung von Prozessen [KoFe20].

Für die Automatisierung von Prozessen kommt zunehmend Robotic Process Automation (RPA) Technologie zum Einsatz [HoSU20], [Eise19]. RPA bietet den Vorteil, dass bereits eine Vielzahl an Werkzeugen und Schnittstellen vorhanden sind, wodurch einer Automatisierung ganzer Geschäftsprozesse technisch nichts im Wege steht. Zudem kann die RPA-Software einfach und mit einer relativ kurzen Implementierungszeit in die bestehende IT-Landschaft integriert werden und es sind oftmals keine tiefgreifenden Programmierkenntnisse für die Erstellung von Robotern erforderlich [LaTu20]. Dadurch werden Roboter auch durch Fachexpertinnen und -experten oder operative Einheiten implementiert, da hier entscheidendes Fachwissen sowie umfangreiches Wissen über die Geschäftsprozesse vorhanden sind.

Bei der Softwareentwicklung werden fachliche Tests zum überwiegenden Teil von MitarbeiterInnen aus den jeweiligen Fachbereichen oder speziellen Testern manuell durchgeführt. Für solche Tests werden zwar Werkzeuge zur Unterstützung und Erleichterung dieser manuellen Tests zur Verfügung gestellt, aber deren Einsatz ist mit einem entsprechenden Lernaufwand wie auch Kosten verbunden. Damit sich die Testautomatisierung in diesem Bereich rechnet, muss es zu einer Trennung von fachlicher Logik und der technischen Implementierung kommen, da dadurch die Erstellung eines automatisierten Testfalles wie auch die Wartung vereinfacht wird [BBSG15].

Da mit RPA diese Trennung zwischen fachlicher Logik und technischer Implementierung vor allem durch die Lowcode-Programmierung verwirklicht wird,

folgen daraus durchaus Synergieeffekte, welche den Einsatz von RPA für die Testautomatisierung rechtfertigen. Solche Synergieeffekte sind einerseits, dass auch bei der Testautomatisierung die Testfälle ein umfangreiches fachliches Verständnis der Prozesse erfordert, was durch den Einsatz von RPA bei der Testautomatisierung mit einem besseren Einbinden von Fachexpertinnen und -experten bzw. auch mit der Beauftragung zur Testfallerstellung der Fachabteilungen gelöst werden kann. Das ist dadurch möglich, weil durch die zum Teil auf grafischem Workflow basierende Lowcode-Programmierung der Roboter keine aufwendige Programmierung bzw. technische Implementierung erforderlich ist. Andererseits kann das Wissen und die Erfahrung von der Entwicklung und der Implementierung eventuell bereits vorhandener Roboter im Unternehmen genutzt werden und es muss keine neue Testautomatisierungssoftware angeschafft und im Unternehmen entsprechend ausgebildet werden. Dies führt wiederum zu einer Zeit- und Kostenersparnis.

RPA-Tools bieten zwar alle Werkzeuge und Schnittstellen für einen End-to-End-Test und durch die zum Teil grafische und einfache Erstellung der Prozesse sind sie gut geeignet für die Zusammenarbeit mit Fachexpertinnen und -experten und Personen, welche von Programmierung keine bis wenig Erfahrung haben. Aber diese RPA-Tools haben keine geeignete Möglichkeit für die Abarbeitung von Test-Cases sowie das Reporting am Ende. Im Detail sind die aktuellen Problemstellungen die Steuerung der einzelnen Roboter-Prozesse bzw. der Abarbeitung der Test-Cases bei einer hohen Anzahl an Test-Cases sowie ein geeignetes Reporting des Testfortschrittes und der Ergebnisse. Gerade bei einer Vielzahl von Prozessen, die mit RPA abgearbeitet werden, braucht es eine Möglichkeit, diese spezifisch für einzelne Bereiche auszuwählen und abarbeiten zu können. RPA bietet nur die Möglichkeit, einzelne Prozesse abzuarbeiten. Das entspricht immer einem Test-Case. Daher braucht es hier die Möglichkeit, mehrerer dieser Prozesse hintereinander zu starten und zu überwachen. Zudem muss auch sichergestellt werden, dass bei einem Fehler im RPA-Prozess der Test nicht abgebrochen wird, sondern zum nächsten Test-Case übergegangen wird. Weiters braucht es bei der Testautomatisierung am Ende jedes Test-Cases eine entsprechende Übersicht der Ergebnisse. Gerade bei Regressionstests, wo bereits auf dem Weg zum entsprechenden Testobjekt Fehler auftreten können, müssen auch einzelne Fortschritte im Prozess dokumentiert werden, damit am Ende auch Fehler schnell nachvollzogen werden können.

Daher beschäftigt sich die Arbeit mit der Konzeption und prototypischen Implementierung eines Frameworks für die Testautomatisierung mit bestehenden RPA-

Tools bei Regressionstests im End-to-End–Bereich, wodurch eine Steuerung und Auswahl der einzelnen Test-Cases sowie ein Reporting der Ergebnisse ermöglicht werden soll.

1.2 Theoretische Grundlagen

Neben der Automatisierung von Geschäftsprozessen gibt es bereits Ansätze, RPA zur Automatisierung von Softwaretests einzusetzen. Gerade bei Graphical-User-Interface-Tests von Web-Applikation oder bei End-to-End-Tests hat sich gezeigt, dass RPA durchaus eine geeignete Alternative ist [CeSS20]. Es gab bereits wissenschaftliche Arbeiten, in denen untersucht wurde, ob RPA bei Web-Applikationen eine geeignete Alternative zu Selenium[1] ist. Dabei wurde festgestellt, dass RPA durchaus für solche Tests eingesetzt werden kann [Heis21]. Weitere wissenschaftliche Untersuchungen zeigten, dass RPA nicht nur bei einfachen Testszenarien, sondern auch bei komplexeren End-to-End-Tests eingesetzt werden kann. Dabei waren die größten Vorteile die schnelle Erstellung von Robotern ohne Programmierkenntnisse. Das bedeutet, nicht nur Programmierer*innen könnten RPA für Testzwecke einsetzen, sondern auch Fachexperten. Man kam aber auch zur Erkenntnis, dass es noch weitere Forschung sowie Anpassung von RPA braucht, damit diese Technologie nicht nur auf einzelne Testszenarien, sondern als primäres Testtool eingesetzt werden kann. So stellt sich die Frage, wie eine geeignete Testinfrastruktur für die Testautomatisierung mit RPA gestaltet sein muss. Dazu zählen beispielsweise Bereiche für Test-Managing, Test-Reporting, Test-Durchführung oder eine Kombination aus diesen [CeSS20].

Bei dieser Arbeit sollen durch die Entwicklung eines Frameworks die RPA-Funktionen für die Testautomatisierung erweitert werden. Frameworks bieten die Möglichkeit, Konzepte und entsprechende Programmcodes in Software-Projekten wiederzuverwenden. Somit ist ein Framework eine abstrakte Architektur, welche Teile der Funktionalität zur Verfügung stellt, aber auch als Referenzmodell dient [Diet02]. Dadurch haben Frameworks den Vorteil, dass sie Entwicklerzeit und -kosten einsparen, da diese Konzepte und Methoden einfach in Software-Projekte implementiert werden können und so des Öfteren zum Einsatz kommen. Zudem vereinfachen Frameworks auch die Entwicklung von Software-Projekten. Durch Frameworks werden komplexe Vorgänge durch Abstraktion verborgen, wodurch eigene Projekte schlanker und übersichtlicher werden [Jäge08]. Dieses Vorgehen ist auch beim Einsatz von RPA-Tools

[1] https://www.selenium.dev/

üblich. So werden einem vor allem für Python unterschiedliche Open Source Frameworks angeboten, welche unterschiedliche Funktionalitäten für die Ver- und Bearbeitung von Daten oder den Zugriff auf andere Anwendungen ermöglichen [Rpaf22].

Die Entwicklung dieses Frameworks wird im Zuge der Arbeit primär in der Programmiersprache C# erfolgen. C# ist eine objektorientierte und moderne Programmiersprache aus der C-Sprachfamilie von Microsoft [Bill22].

Als theoretische Grundlage dieser Arbeit dienen bereits bestehende Vorgehensweisen bei Softwaretests und Testframeworks. Damit sollen die Anforderungen an ein Framework für den Einsatz von RPA für die Testautomatisierung und der aktuelle Stand der Technik bzw. der Forschung ermittelt werden. Zudem wird die Funktionsweise von Open Source RPA Tools untersucht und die Vorgehensweise beim Automatisieren von Prozessen mittels RPA beleuchtet. Daraus sollen Erkenntnisse gewonnen werden, wie ein Framework für den Einsatz von RPA in der Testautomatisierung aus der Sichtweise von RPA konzipiert sein könnte.

Die Auswahl der RPA-Tools hinsichtlich der Anforderungsanalyse in Bezug auf die Anknüpfungspunkte des Frameworks für diese Arbeit soll sich auf jene RPA-Tools beschränken, welche entweder als Open Source oder kostenlos zur Verfügung stehen. Alternativ können auch jene RPA-Tools einbezogen werden, welche online eine ausführliche Beschreibung über die Schnittstellen zur Verfügung stellen. Für die Umsetzung des Prototyps soll OpenRPA oder taskt zum Einsatz kommen. Beide RPA-Tools sind Open-Source-Lösungen und bieten eine Vielzahl an Funktionen für die Automatisierung.

1.3 Arbeitsziel

Das Ziel dieser Arbeit ist es herauszufinden, wie mit RPA eine geeignete Testautomatisierung in Bezug auf die Steuerung der einzelnen Test-Cases sowie für das Test-Reporting möglich ist. Mit dieser Arbeit sollen die offenen Fragen auf die Implementierung von RPA in eine Testinfrastruktur beantwortet werden und so ein Beitrag an die wissenschaftliche Gemeinschaft geliefert werden. Zudem sollen mit dem Framework nicht nur die funktionalen Anforderungen abgedeckt werden, sondern auch eine Referenzarchitektur für den Einsatz von RPA zur Testautomatisierung entwickelt werden.

Im Detail sollen Erkenntnisse darüber erlangt werden, welche Schnittstellen bei den gängigen RPA-Tools vorhanden sind, um das Framework einzubinden - wie die Informationsübertragung zwischen Framework und RPA-Tool funktionieren kann. Welche Informationen werden am Ende eines Testdurchlaufes für die Erstellung einer Übersicht der Ergebnisse benötigt. Wie kann diese Übersicht am Ende erstellt werden und wie sollen die Informationen darin dargestellt werden.

1.4 Forschungsfrage

Die Arbeit soll folgende Forschungsfrage beantworten:

„Wie muss ein Framework für den Einsatz von Robotic Process Automation Tools zur Testautomatisierung bei Regressionstests im End-to-End-Bereich konzipiert sein, damit die Steuerung der einzelnen Test-Cases in Form von RPA-Prozessen sowie ein Test-Reporting der Ergebnisse ermöglicht wird?“

1.5 Methodische Vorgangsweise

Zur Beantwortung der Forschungsfrage kommt als methodischer Rahmen das Design Science Research von Hevner et al. zum Einsatz. Dieser methodische Rahmen ermöglicht es, durch strukturiertes Vorgehen ein Design Artefakt zu erstellen. Ein Design Artefakt kann je nach Erfordernis eine Vielzahl an Formen annehmen. Dabei kann es sich beispielsweise um eine Methode, ein Modell, ein Framework, einen Prototyp oder um ein Softwareprodukt handeln. Für die Gestaltung dieses Artefakts beschreibt Hevner et al. sieben Richtlinien [HMPR04].

1. **Artefaktgestaltung**: Design Science Research hat als Ziel ein echtes Artefakt mit einem praktischen Nutzen.
2. **Problemrelevanz**: Das Ergebnis muss ein relevantes Problem lösen.
3. **Designevaluation**: Um den Nutzen, die Qualität und die Wirksamkeit des Artefakts bewerten zu können, müssen Evaluationsmethoden eingesetzt werden.
4. **Forschungsbeitrag**: Es muss neben der Lösung des Problems auch ein allgemeingültiger Forschungsbeitrag geleistet werden.
5. **Forschungsstrenge**: Es müssen bei der Forschung strikte Methoden für die Konstruktion und Evaluation eingesetzt werden.
6. **Design als Optimierungsproblem**: Bei der Suche nach der optimalen Lösung sollen alle verfügbaren Mittel unter Einhaltung der Rahmenbedingungen der Problemumgebung eingesetzt werden.

7. **Publikationsfähigkeit**: Veröffentlichung der Forschungsergebnisse bzw. des Artefakts.

Dabei handelt es sich um eine sehr praxisnahe Vorgehensweise. Das Design Science Research nach Österle et al. ist eine Ableitung aus dem Modell von Hevner et al., welches einen vierstufigen Erkenntnisprozess vorschlägt [BeKS20]. Das Modell von Österle et al. ist eine gestaltungsorientierte pragmatische Vorgehensweise, welche bei dieser Arbeit zum Einsatz kommen soll. Das strukturierte Vorgehen bei dieser Methode erfolgt in den 4 Iterationsschritten, Analyse, Entwurf, Evaluation und Diffusion sowie der Evaluierung, welche auch die oben beschriebenen Richtlinien von Hevner et al. beinhalten [PTRC07].

In Tabelle 1 ist ersichtlich, wie sich diese Richtlinien nach Hevner et al. in den einzelnen Iterationsschritten von Österle et al. wiederfinden.

Iterationsschritte nach Österle et al.	Richtlinie nach Hevner et al.
Analyse	Problemrelevanz
Entwurf	Artefaktgestaltung Forschungsbeitrag Forschungsstrenge Design als Optimierungsproblem
Evaluation	Designevaluation
Diffusion	Publikationsfähigkeit

Tabelle 1: Zuordnung der Richtlinien an die Iterationsschritte [Zwac17]

Die Methode des Design Science wurde deshalb für diese Arbeit gewählt, da die Anforderungen an das Framework unbekannt sind und mit dieser Methode eine Vorgehensweise von der Analyse bis zur Diffusion stattfindet. Zudem kann durch die jeweiligen Iterationen zwischen den einzelnen Schritten das Framework Schritt für Schritt erarbeitet, aber auch neue Erkenntnisse aus einem Folgeschritt wieder durch Rücksprünge in den vorigen Schritt miteingearbeitet werden. Abbildung 1 soll diese Vorgehensweise mit den Ergebnissen der jeweiligen Iterationsschritte bei dieser Arbeit etwas näher erläutern. Dieses Vorgehen sowie diese vier Iterationsschritte spiegeln sich auch im Aufbau bzw. in der Struktur der Arbeit wider.

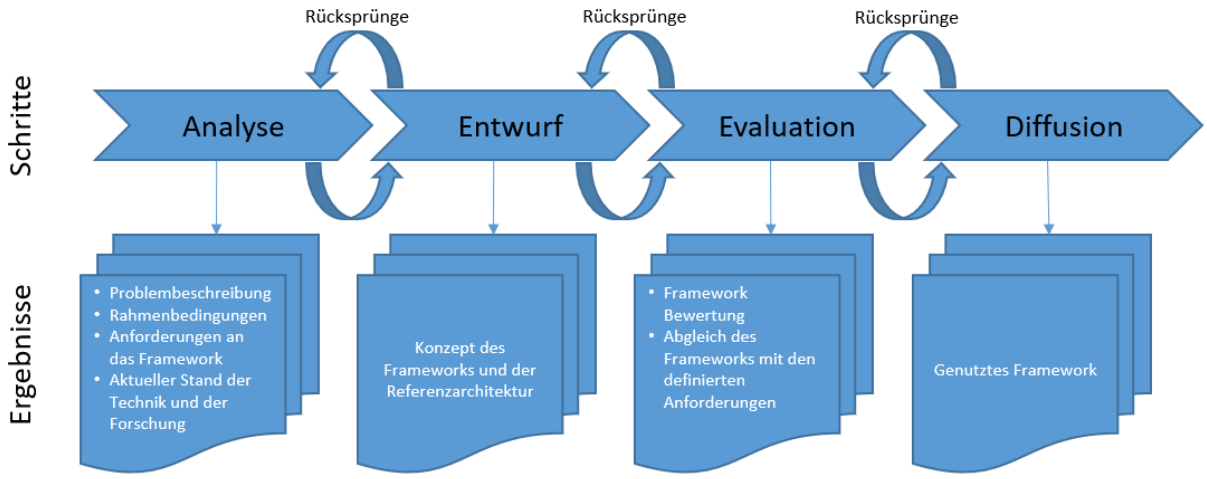


Abbildung 1: Vorgehensweise bei dieser Arbeit (eigene Darstellung in Anlehnung an [BeKS20])

Analyse

Das bei der Analyse beschriebene Ergebnis der Problembeschreibung umfasst bereits das Kapitel 1.1 dieser Arbeit. Die weiteren zwei Ergebnisse, Rahmenbedingungen und Anforderungen an das Framework, sollen sich aus dem dritten Ergebnis, der aktuelle Stand der Technik und Forschung, ableiten lassen. Hierfür soll die Methodik der Literaturrecherche zum Einsatz kommen, wo eben die Grundlagen dieser, aktueller Stand der Technik und Forschung, mit Schwerpunkt zu den Bereichen Testautomatisierung und RPA ausgearbeitet wird. Das ist wichtig, damit die Anforderungen und Rahmenbedingungen für den zweiten Schritt vollständig und richtig sind, da hier aufbauend auf dem ersten Schritt das Konzept und die Referenzarchitektur für das Framework entworfen wird.

Entwurf

Beim zweiten Schritt, der Entwurf, erfolgt auf Basis des ersten Schrittes die Konzeption des Frameworks. Um hier ein geeignetes Konzept für ein Framework entwickeln zu können, kommt die Methode der Referenzmodellierung zum Einsatz. Diese Methode hat zum Ziel, eine abstrakte Abbildung der Realität wiederzugeben [WiHe06]. Für diesen Zweck können unterschiedliche Modellierungssprachen mit vordefinierten Notationen zum Einsatz kommen. So können Prozesse, Datenstrukturen sowie Klassenmodelle abgebildet werden. Das Ziel eines solchen Referenzmodells ist es, einerseits sicherzustellen, dass die Anwendung oder Prozesse die Anforderungen optimal unterstützt bzw. fehlerfrei umgesetzt werden und andererseits sollen die auf dem Referenzmodell aufbauenden Systeme oder Teilsysteme schneller und effizienter umgesetzt werden können [WaAL07]. Bei der Arbeit werden Modelle wie das Unified

Modeling Language (UML) Klassendiagramm oder erweiterte ereignisgesteuerte Prozessketten (eEPK) zum Einsatz kommen. Zudem sollen bei den einzelnen Modellen immer die Anforderungen angeführt werden, welche mit den jeweiligen Modellen abgedeckt werden. Sollten Rücksprünge erforderlich sein, werden hier auch die Gründe angeführt und beschrieben.

Evaluation

Im dritten Schritt wird auf Basis des Konzeptes das Framework programmiert und bewertet. Hierfür kommt die Methode des Prototypings zum Einsatz. Bei einem Prototyp muss es sich nicht um ein komplettes System handeln, sondern um ein Teilsystem aus einem Gesamtsystem sowie die damit verbundenen vorgelagerten Arbeiten zur Implementierung einer Software [RoSt20]. In der Arbeit soll ein Funktionsprototyp eingesetzt werden. Damit soll das zuvor konzipierte Framework implementiert werden und in weiterer Folge soll anhand dieses Prototyps evaluiert werden, ob die konzipierten Anforderungen an das Framework umgesetzt und diese geeignet sind, um eine entsprechende Steuerung der einzelnen Test-Cases sowie ein Test-Reporting zu ermöglichen. Die Implementierung des Prototyps erfolgt im Zusammenhang mit einer praxisorientierten Aufgabenstellung, bei der eine Software mittels Testautomatisierung durch RPA getestet werden soll. Durch diese praktische Implementierung stehen geeignete echte Test-Cases zur Verfügung, anhand dieser eine Evaluierung des Prototyps ermöglicht wird. Am Ende soll eine Evaluierung des Prototyps durchgeführt werden. Bei dieser Evaluierung soll die Funktionsweise auf ihre Richtigkeit geprüft, ob das Framework den Anforderungen und den Rahmenbedingungen entspricht sowie eine kritische Betrachtung durch das Aufzeigen der Vor- und Nachteile des Frameworks durchgeführt werden. Auch hier sollen eventuelle Rücksprünge begründet und dokumentiert werden.

Diffusion

Im vierten und letzten Schritt wird die Verwendung bzw. die zur Zurverfügungstellung des Frameworks diskutiert. Hier sollen die weiteren Vorgehensweisen wie Schulungen und Prozessanpassung bei der Implementierung in einem Unternehmen diskutiert und aufgezeigt werden. Zudem soll auch hier die Zurverfügungstellung als Open-Source-Projekt und deren Nutzen diskutiert werden.

Diese theoretische Auseinandersetzung mit dem Design Science Research in Verbindung mit der Thematik der Arbeit zeigt, dass dieses Vorgehen eine geeignete Methode darstellt, um die Forschungsfrage zu bearbeiten.

2 Grundlagen und Stand der Technik

Dieses Kapitel umfasst den theoretischen Hintergrund dieser Arbeit. Hierzu wird zunächst im Abschnitt 2.1 die Testautomatisierung bei Softwaretests näher beleuchtet. Es erfolgt eine Betrachtung der Prozesse bei der Testautomatisierung und eine Beschreibung der Framework-Architektur bei der Testautomatisierung. Die Testarten End-to-End-Test und Regressionstest sowie die unterschiedlichen Möglichkeiten zur Ausführung von Regressionstests werden beschrieben. Zusätzlich werden die allgemeinen und die in Bezug auf die Regressionstests erforderlichen Bestandteile des Test-Reports hinsichtlich Informationsgehalt und Darstellung näher betrachtet. Im Abschnitt 2.2 erfolgt die Beschreibung von RPA. Auch hier wird auf die Architektur, Funktionsweise sowie auf die Einsatzgebiete näher eingegangen. Am Ende erfolgt eine Beschreibung aktueller RPA-Tools. Dabei wird ein Vergleich der Funktionen, welche für den Einsatz von RPA zur Testautomatisierung erforderlich sind, erstellt.

2.1 Beschreibung Testautomatisierung bei Softwaretests

Unzuverlässige oder inkorrekte Software kann zu enormen Problemen wie etwa Verlust von Zeit und Geld, Rufschädigung bis hin zu Personenschäden führen. Daher ist es wichtig, vor einem Deployment die Qualität einer Software zu prüfen. Dadurch kann das Risiko eines Fehlers oder sogar der Ausfall einer Software reduziert werden [SpLi19]. Bei neuen Software-Systemen steigt zunehmend die Größe und die Komplexität. Vor allem die zunehmende Anzahl an verschiedenen Komponenten und die Verstrickungen untereinander und zur Umwelt sind wesentliche Komplexitätstreiber [SnJu11]. Das führt dazu, dass Software-Systeme nur durch einen enorm hohen Ressourcenaufwand getestet werden können, da für eine gute Testabdeckung eine hohe Anzahl an Testfällen benötigt wird [AABK05]. Nur durch eine entsprechend hohe Testabdeckung ist es möglich, eine hohe Anzahl an Fehlern noch vor einem Deployment zu entdecken.

Da der Einsatz von Software meist wesentlich günstiger ist als die menschliche Arbeitszeit, werden solche Tests automatisiert oder, wenn bestimmte Eingriffe vom Personal erforderlich sind, auch teil-testautomatisiert durchgeführt. Zudem können automatisierte Tests schneller und auch zu jeder Zeit ausgeführt werden. Unter

Testautomatisierung im Zusammenhang mit Softwaretests versteht man, dass manuelle durch den Menschen ausgeführte Testaktivitäten ohne Eingriff eines Menschen von einer Maschine abgearbeitet werden [AABK05].

2.1.1 Testprozess

Softwaretests sollten bereits zu einem sehr frühen Zeitpunkt im Entwicklungsprozess beachtet werden [BBSG15]. Zudem gibt es beim Testen wie auch bei der Softwareentwicklung Vorgehensweisen und Aktivitäten, die je nach Projekt durchlaufen werden müssen. Abbildung 2 zeigt den fundamentalen Testprozess mit den einzelnen Testaktivitäten, welcher nach dem Lehrplan von International Software Testing Qualifications Board (ISTQB) Certified Tester vorgeschlagen wird [BBSG15]. Dieser Testprozess dient als Leitfaden und soll bei jeder Testart von Modultest bis zum Systemtest eingehalten werden [Witt19].

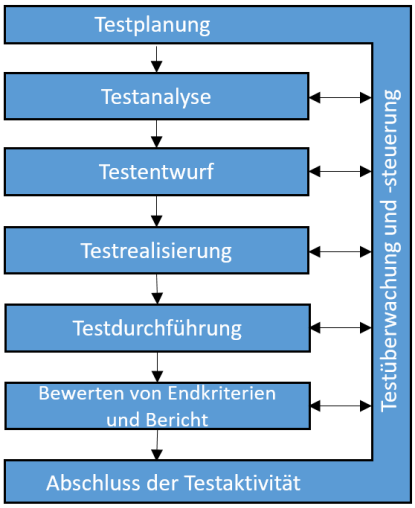


Abbildung 2: Testprozess nach dem Lehrplan des ISTQB Certified Tester [BBSG15]

Testplanung, Testüberwachung und -steuerung

Am Beginn des Testprozesses steht die Testplanung. Hier werden die Rahmenbedingungen für die Testaktivitäten festgelegt. Der Output der Testplanung ist das Testkonzept [BBSG15]. In diesem Konzept werden die Ressourcenplanung, die Teststrategie, die Testpriorisierung sowie das Testwerkzeug beschrieben. Zusätzlich ist neben einem Zeitplan auch noch eine entsprechende Aufwandsschätzung enthalten. Zudem fließen die Ergebnisse aus der Testfallüberwachung der einzelnen Testaktivitäten in die Planung mit ein [AABK05].

In der Testplanungsphase muss bereits mit der Beurteilung begonnen werden, welche Testfälle automatisiert durchgeführt werden sollen. Für diese Beurteilung werden folgende Kriterien herangezogen [BBSG15]:

- Je höher die Anzahl der Wiederholung von Testfällen, umso höher die Einsparungen von Zeit und Kosten.
- Umso länger die Entwicklungs- und Lebensdauer von Applikationen, umso profitabler ist die Automatisierung.
- Bei Testfällen, welche mit einer hohen Anzahl an unterschiedlichen Daten getestet werden können, kann durch die Automatisierung ein höherer Anteil an Datenkombinationen getestet werden.
- Automatisierte Testwiederholungen führen die Tests immer in derselben Qualität aus.
- Bei Regressionstests muss mit der Testautomatisierung eine entsprechende Sicherheit vorhanden sein, damit Funktionalitäten bei Änderungen am Testobjekt erkannt werden.

Testanalyse, Testentwurf

Ausgangslage für diese Phasen ist die Testbasis. Die Testbasis bezeichnet alle Dokumente oder entsprechendes Fachwissen, welches für die Beurteilung eines Fehlverhaltens in der Software herangezogen werden kann. Es beschreibt somit das Sollverhalten des Testobjektes [SpLi19]. Darauf aufbauend werden die logischen Testfälle entworfen.

Zudem werden in dieser Phase die Testfälle für die Testautomatisierung umgesetzt. Zu Beginn müssen die geplanten Testfälle für die Testautomatisierung auf ihre technische Machbarkeit geprüft werden, da nicht immer alle Testfälle in einem vertretbaren Aufwand umsetzbar sind. Eventuell müssen Testfälle abgewandelt werden, damit sie mit den verfügbaren Mitteln umgesetzt werden können. Bei der Testautomatisierung ist auf eine breite Testabdeckung zu achten. Testfälle, die nicht zur Gänze automatisch durchlaufen werden können und ein Eingreifen des Menschen erfordern, sollten nicht zur Testautomatisierung gezählt werden, sondern das manuelle Testen des Menschen unterstützen [BBSG15].

Testrealisierung

Die bereits logisch erstellten Testfälle werden mit entsprechenden Testdaten hinterlegt. Sobald das Testobjekt die jeweiligen Testfälle zur Verfügung stellt, werden die Testfälle abgearbeitet [BBSG15].

In der Phase der Testrealisierung beginnt die technische Implementierung der Testfälle mit einem entsprechenden Testautomatisierungswerkzeug. Dabei entsteht für jeden Testfall ein Testskript, welches aus Prozessinformationen und Testdaten besteht. Bei der Implementierung kann zwischen den zwei Arten Capture Replay und dem Einsatz eines Testframeworks unterschieden werden. Bei Capture Replay wird in einem ersten Schritt der Pfad eines Testfalles zu Masken, Reitern und Eingabeelementen von einem Testobjekt durch einen*eine Tester*in in Form eines Testskripts aufgezeichnet. Die Aufzeichnung erfolgt dabei automatisch durch ein entsprechendes Capture Replay Tool. Im nächsten Schritt erfolgt eine Trennung der Prozessinformationen von den Testdaten, indem die Eingaben durch Variablen ersetzt werden. Dadurch ist es möglich, dieses Testskript mit einer Vielzahl unterschiedlicher Testdatenkonstellationen durchlaufen zu können [Jack09]. Beim Einsatz von Testframeworks ist hingegen mehr Programmiertätigkeit erforderlich, aber im Gegenzug ist man bei der Verwendung von Testframeworks flexibler bei Änderungen am Testobjekt und beim Anpassen der Testfälle. Hierbei erfolgt ein modularer Aufbau der Testskripte. Module bzw. Funktionsbausteine werden in entsprechenden Bibliotheken eines Testframeworks zusammengefasst. Diese können dann bei Testfällen entsprechend zusammengesetzt und erweitert werden. Zudem ist es sinnvoll, die Bibliothek in mehreren Projekten zu verwenden und entsprechend zu erweitern. Somit stehen Testskripte langfristig zur Verfügung und der Testautomatisierungsaufwand kann bei neuen Projekten reduziert werden [EbSt11].

Unabhängig davon, wie die Testskripte erstellt werden, können sich auch hier Fehler einschleichen. Daher sollten hier entsprechende qualitätssichernde Maßnahmen eingehalten werden. In dieser Phase kann auch festgestellt werden, dass einige Testfälle technisch nicht für die Testautomatisierung geeignet sind. Am Ende müssen die umgesetzten Testfälle dahingehend überprüft werden, ob sie ihren Zweck erfüllen und reibungslos funktionieren [BBSG15].

Testdurchführung

Bei der Testautomatisierung ist für die Rückverfolgung von Fehlern eine entsprechende Testprotokollierung erforderlich. Für jeden Testzyklus sind eigene Protokolle anzulegen. Die Protokollierung kann auf unterschiedliche Art je nach Erfordernis erfolgen. Mögliche Protokolle sind Schrittprotokollierung, Fehlerprotokollierung, Videos oder Screenshots. Für eine spätere Auswertung dieser Protokolle ist eine strukturierte Ablage erforderlich [BBSG15].

Bewerten von Endkriterien und Bericht

Nach der Testdurchführung werden die Testergebnisse den Sollkriterien gegenübergestellt. Abweichungen zwischen den erwarteten und den tatsächlichen Ergebnissen müssen dem*der Tester*in in einer verständlichen Form aufbereitet und dargestellt werden [ArOS04]. Die Gegenüberstellung hat beim jeweiligen Testobjekt als Ergebnis, dass der Test entweder den Erfordernissen entspricht oder dass entsprechende Nacharbeiten erforderlich sind [BBSG15].

Abschluss der Testaktivität

In dieser Phase erfolgt ein Rückblick auf die jeweiligen Testaktivitäten. Dabei stehen hier die Dokumentation von Best-Practice und die Überprüfung im Fokus, ob die Ziele erreicht wurden. Zudem werden die Testmittel archiviert, damit sie bei einer Wiederholung der Testaktivitäten wieder zur Verfügung stehen [BBSG15].

2.1.2 Teststufen

Für das Testen komplexer Softwaresysteme werden die Tests nach mehreren Stufen unterteilt. Diese richten sich nach dem jeweiligen Entwicklungsstand des Systems. Grundsätzlich gilt, je früher getestet wird, umso besser, da das frühzeitige Erkennen von Fehlern viel Zeit und Geld spart. Aber nicht in jeder Entwicklungsphase stehen einem alle Funktionen, Module, Komponenten oder Schnittstellen zur Verfügung. Daher ist beim Testen zwischen den einzelnen Teststufen eine Abgrenzung zu machen und zu prüfen, welcher Test in welchem Stadium der Softwareentwicklung sinnvoll

durchgeführt werden kann [Witt19]. Abbildung 3 zeigt eine Schematische Darstellung der Teststufen mit dem Zeitbedarf der einzelnen Testabläufe und den Testinhalt.

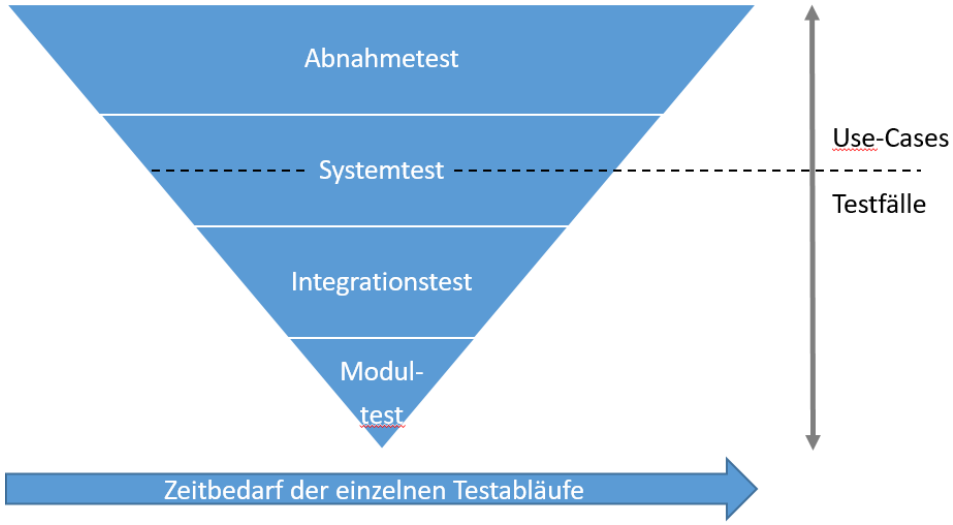


Abbildung 3: Teststufen mit dem Zeitbedarf (eigene Darstellung in Anlehnung an das V-Modell von [Witt19])

Bei einem Modultest werden nur einzelne Funktionen, Komponenten oder Module getestet. Damit soll sichergestellt werden, dass die Testergebnisse nicht durch andere Ereignisse beeinflusst werden [Hard00]. Diese Tests können bereits zu einem sehr frühen Stadium der Softwareentwicklung eingesetzt werden und sie werden durch die Entwicklerteams durchgeführt [BBSG15].

Der Integrationstest hat zum Ziel, das Zusammenspiel zweier oder mehrerer Komponenten zu testen. Diese Komponenten können aus einzelnen Softwarebausteinen oder Teilsystemen bestehen. Dabei wird festgestellt, ob der Datenaustausch den Erfordernissen entspricht oder die Schnittstelle korrekt implementiert wurde [Hard00].

Beim Systemtest steht das Gesamtsystem im Fokus der Tester. Dabei soll die spezifikationsgemäße Funktionsweise des gesamten Systems getestet werden [AABK05]. Der Test des Gesamtsystems erfolgt hier immer aus der Sicht der Auftrag gebenden Person bzw. des Anwendenden [Hard00]. Diese Tests werden zum Teil anhand konkreter Use-Cases durchgeführt. Dabei sollen möglichst viele Nutzungsabläufe bei den Testabläufen einbezogen werden [HoRK08]. Bei der Testautomatisierung sind hier oft die einzigen Schnittstellen die Datenbank oder die Benutzeroberfläche der Software, welche nicht für den Zugriff eines anderen Programmes konzipiert ist [BBSG15]. Die Testumgebung soll beim Systemtest so gut wie möglich der Produktivumgebung entsprechen [Witt19].

Die letzte Stufe ist der Abnahmetest. Bei diesem Test werden die zu Beginn festgelegten Anforderungen bei der erstellten Software abgenommen [BBSG15]. Ziel dieses Tests ist es nicht mehr, Fehler aufzudecken, sondern der Kunde oder die Kundin soll Vertrauen in die neue Software oder zu den neuen Funktionalitäten gewinnen. Diese Tests umfassen eine geringe Testabdeckung, aber dafür beinhalten die Testabläufe die wichtigsten Use-Cases bzw. typische Geschäftsprozesse, welche im Zusammenhang mit der Software abgedeckt werden sollen [Witt19].

2.1.3 Framework-Architektur bei der Testautomatisierung

Abbildung 4 zeigt den prinzipiellen Aufbau einer Framework-Architektur bei der Testautomatisierung. Ein zentrales Merkmal eines Automatisierungssystems ist die Unabhängigkeit zwischen Testfällen, den Werkzeugen und dem Testobjekt [BBSG15].

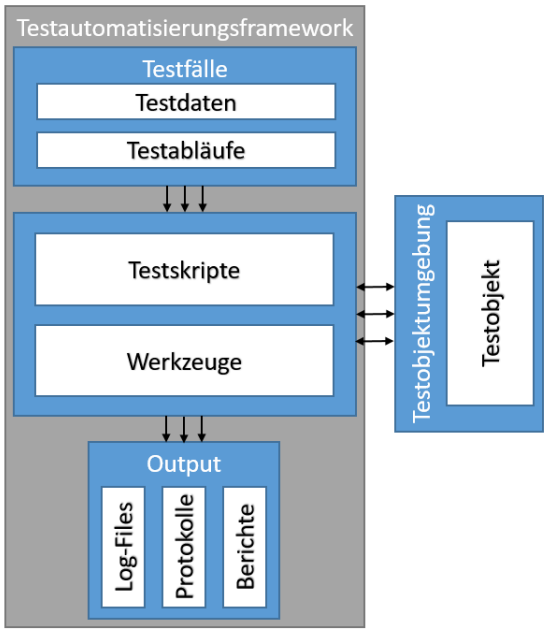


Abbildung 4: Framework-Architektur bei der Testautomatisierung (eigene Darstellung in Anlehnung an das Testautomatisierungsframework von [BBSG15] und die Testautomationslandschaft von [EbSt11])

Als Basis für die Testautomatisierung werden in erster Linie Testfälle benötigt. Diese bestehen aus den Testdaten und den Testabläufen [BBSG15]. Damit bei jedem Testzyklus die Ergebnisse vergleichbar sind, müssen die Bedingungen, unter denen der Testfall abgearbeitet wird, gleichbleiben. Daher ist es erforderlich, die Testdaten zentral und dauerhaft zu verwalten [Witt19]. Testabläufe enthalten die fachlichen Informationen, welche für die Durchführung eines Tests notwendig sind [SpLi19]. Für die Beschreibung der Testabläufe werden einheitliche Templates verwendet. Mit solchen Templates wird sichergestellt, dass der Aufbau der Informationen immer der gleiche ist

und jeder Mitarbeiter und jede Mitarbeiterin die Struktur der Testablaufbeschreibung kennt. Zudem müssen die Testabläufe so beschrieben sein, dass auch Mitarbeiter und Mitarbeiterinnen, welche mit dem Testobjekt nicht vertraut sind, den Test durchführen bzw. die Ergebnisse nachvollziehen können. Um ein Testobjekt ausführlich testen zu können, sind meist mehrere solcher Testabläufe erforderlich. Dabei kann es auch zwischen den einzelnen Testabläufen Abhängigkeiten geben. Das ist der Fall, wenn ein Testablauf einen bestimmten Status oder eine Voraussetzung des Testobjektes erfordert [Witt19]. Je nach Testart müssen ganze Geschäftsprozesse getestet werden. Diese müssen zerlegt und bis auf Arbeitsanweisungen heruntergebrochen werden [Oste16].

Testskripte sind technische Aktionen, welche in der Reihenfolge wie im Testablauf beschrieben, abgearbeitet werden. Dabei werden die Aktionen über verschiedenste Werkzeuge auf dem Testobjekt ausgeführt. Diese Testskripte sind meist modular aufgebaut und sie bedienen sich je nach Einsatzbereich unterschiedlicher Werkzeuge und Bibliotheken. Bei der Erstellung von Testskripten muss vor allem auf die Wartbarkeit Bedacht genommen werden, damit Änderungen durch den Austausch von Modulen oder Funktionen den Testfall wieder lauffähig machen [BBSG15]. Damit auf das Testobjekt zugegriffen werden und damit interagiert werden kann, sind bestimmte Werkzeuge erforderlich. Dabei kann der Zugriff über bestimmte Schnittstellen wie Datenbanken, Webservice oder die Benutzeroberfläche erfolgen. Bei der Benutzeroberfläche müssen diese Werkzeuge gezielt Elemente wie Buttons, Textboxen oder Tabellen ansteuern und Aktionen darauf ausführen können [Witt19]. Vor allem bei einer Benutzeroberfläche einer Applikation gibt es keine technische Schnittstelle, welche gezielt programmiertechnisch angesteuert werden kann. Hier nimmt das Werkzeug die Rolle einer technischen Schnittstelle zum Testobjekt ein [BBSG15].

Das Testobjekt ist in eine Testobjektumgebung eingebettet. Die Testobjektumgebung versetzt das Testobjekt in den erforderlichen testbaren Zustand [KoBe10]. Dieser Zustand soll der späteren Produktivumgebung ähneln. Daher sollen auch in der Testobjektumgebung möglichst die gleichen Hardware- und Softwarekomponenten installiert sein. Die Ausführung der Tests in der tatsächlichen Produktivumgebung sollte auf jeden Fall vermieden werden. Das Testobjekt kann die Produktivumgebung beeinträchtigen, sodass es zu Systemausfällen oder Datenverlust kommen kann [SpLi19]. Das Testobjekt umfasst das zu testende System, bestehend aus verschiedenen Objekten und Funktionen. Bei den Objekten handelt es sich um die Benutzeroberfläche, Datenbanken oder Schnittstellen, auf die beim Testen zugegriffen werden muss. Bei den Funktionen handelt es sich um die Anwendungsfälle, welche aus den

Anforderungsspezifikationen hervorgehen [SnBS09]. Diese Anwendungsfälle spiegeln sich zum Teil in den Testabläufen wider.

Bei der automatisierten Durchführung von Tests entstehen mehrere Log-Dateien, Protokolle und Testergebnisse. Diese dienen einer späteren Fehleranalyse bzw. Rückverfolgbarkeit von Ereignissen [BBSG15]. Dabei muss zwischen Log-Dateien und Protokollen unterschieden werden. Log-Dateien enthalten technische Informationen über einen Ablauf oder ein Ereignis, wohingegen die Protokolle fachliche Informationen enthalten. Aus dem Protokoll muss hervorgehen, welcher Testablauf getestet wurde und welche fachlichen Schritte abgearbeitet wurden und mit welchem Ergebnis [SpLi19]. In einem weiteren Schritt müssen diese Testergebnisse analysiert und zu einem Bericht aufbereitet werden. Dabei ist es wichtig, dass diese Testergebnisse strukturiert abgelegt werden und nach Möglichkeit miteinander verlinkt sind. Zudem muss auch eine eindeutige Zuordnung der Ergebnisse zu den jeweiligen Testabläufen möglich sein, damit die Testabläufe bewertet und eventuell auftretende Fehler im Ablauf zugeordnet werden können [Witt19]. Je nach Testart können sich die Anforderungen an die Inhalte in einem Testbericht unterscheiden. Eine detailliertere Ausführung ist im Abschnitt 2.1.6 vorhanden.

2.1.4 Regressionstest

Bei einem Regressionstest werden gleichbleibende Tests bei einer Änderung am System wiederholt ausgeführt. Dabei können ganze Systeme oder nur Teile getestet werden. Mit solchen Tests soll vermieden werden, dass unerwartete und unerwünschte Nebeneffekte durch Änderungen am Testobjekt entstanden sind, sogenannte Regressionen. Das bedeutet, dass nach Änderungen am Testobjekt beim Test mit demselben Input dasselbe Ergebnis erzielt werden soll und somit die ursprünglichen Funktionen, welche von den Änderungen nicht betroffen sein sollen, unverändert bleiben [Hers00]. Bei Regressionstests ist auf eine sehr hohe Testbreite bis hin zu benachbarten und verbundenen Systemen zu achten. Daher ist bei Regressionstests vor allem ein hoher Wissensstand über die Architektur, die Arbeitsweise des Testobjektes sowie die Prozesse, die damit abgearbeitet werden, wichtig [Witt19]. Solche Regressionstests sollten immer sofort nach jedem Build des Testobjektes ausgeführt werden, damit Fehler frühzeitig erkannt und behoben werden können [HuKo07].

Für den Einsatz von Regressionstests gibt es unterschiedliche Ausprägungen. Abbildung 5 zeigt einen Überblick über die Techniken der Regressionstests.

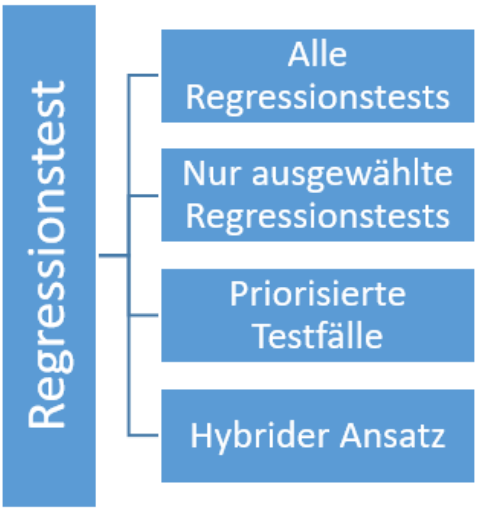


Abbildung 5: Techniken von Regressionstests (eigene Darstellung in Anlehnung an [DuSu08])

Alle Regressionstests

Bei dieser Technik werden alle Testfälle, welche für einen Regressionstest vorgesehen sind, ausgeführt. Das ist bei kleineren Projekten in den meisten Fällen die herkömmliche Methode, um Regressionstests durchzuführen. Diese Technik ist aber sehr zeit- und kostenintensiv [DuSu08]. Daher ist diese Technik bei großen Softwareprojekten eher unpraktisch, da hier die Wirtschaftlichkeit priorisiert wird und während eines Regressionstests die Modifikationen nicht übernommen werden können. Durch das Testen des gesamten Softwaresystems ist diese Technik aber die genaueste [Hers00].

Ausgewählte Regressionstests

Bei dieser Technik wird nur eine Teilmenge aller Testfälle für einen Regressionstest ausgewählt. Dieses Vorgehen spart Kosten und Zeit, führt aber auch dazu, dass bei einer schlechten Auswahl, Graubereiche unter Umständen nicht getestet werden [Hers00]. Eine Möglichkeit zur Selektion von geeigneten Testfällen richtet sich nach der Testabdeckung. Hier wird nach jenen Teilen im Testobjekt gesucht, welche vom Testablauf abgedeckt werden [DuSu08]. Eine andere Selektionsmöglichkeit besteht darin, durch die Modifikationen veraltete Regressionstests wegzugelassen und für zukünftige Regressionstests nicht mehr zu verwenden [Vija07].

Priorisierte Testabläufe

Hier wird jedem Testfall eine Priorisierung zugeordnet. Dabei werden Testfälle mit einer höheren Priorität ausgeführt und niedrigere priorisierte Testfälle werden dabei

weggelassen [DuSu08]. Die Schwierigkeit liegt in der richtigen Priorisierung der Testfälle. Eine Möglichkeit wäre, die Priorisierung mittels der aufgezeichneten Fehlerrate bei vergangenen Testzyklen durchzuführen. Anders wäre es auch möglich, die Priorisierung anhand der geänderten Module oder des Quellcodes durchzuführen [Hers00].

Hybrider Ansatz

Bei dieser Methode werden die zwei Ansätze der Auswahl von Regressionstests und der Priorisierung nach den Testfällen kombiniert. Hier werden nur ausgewählte Regressionstests ausgeführt, welche zusätzlich noch eine Prioritätenreihung besitzen [DuSu08].

Die Auswahl geeigneter Testfälle ist abhängig von den vorhandenen Ressourcen und erfordert immer ein Abwägen zwischen Testabdeckung, Ausführungszeit und dem Aufwand für die Auswahl geeigneter Testfälle. Zudem braucht es auch ausführliche Kenntnisse über das Testobjekt sowie der durchgeführten Modifikationen [Hers00].

2.1.5 End-to-End-Test

Bei einem End-to-End-Test werden die Testabläufe so durchgeführt, dass sie ganze Geschäftsprozesse innerhalb einer Anwendung aus Sicht des Anwendenden abbilden. Solche End-to-End-Tests werden zum Teil auch von Fachexperten und Fachexpertinnen bzw. Business Vertreter*innen durchgeführt. Für diese Art von Test sind keine besonderen technischen Kenntnisse erforderlich, da diese über die Benutzeroberfläche oder Web-Oberfläche des jeweiligen Testobjektes durchgeführt werden und die Testabläufe sich nach den fachlichen Spezifikationen richten. [Sieb00]. End-to-End-Tests werden im Zuge von Systemtests bzw. bei Abnahmetests durchgeführt, da in diesen Phasen sich die Testabläufe auf die Geschäftsprozesse und typischen Anwendungsfälle beschränken und diese aus Sicht des Anwendenden durchgeführt werden [Witt19]. Hierfür werden bei der Testautomatisierung auch spezielle Werkzeuge benötigt, welche über die Benutzeroberfläche als Schnittstelle mit dem Testobjekt interagieren können [BBSG15].

2.1.6 Test-Reporting

Es wird hier nur auf die Anforderungen an das Test-Reporting für den Regressionstest im End-to-End-Bereich näher eingegangen. Das Test-Reporting für einen

Regressionstest muss gut strukturiert und sollte möglichst detailliert sein, damit festgestellte Regressionen schnell lokalisiert und diese entsprechend als Fehler oder Fehlalarm klassifiziert werden können [Hers00]. Das Test-Reporting sollte ohne tiefergehende Kenntnisse in der Testautomatisierung leicht zu lesen und zu verstehen sein, sodass sich auch andere Personen einen schnellen Überblick verschaffen können - im Gegensatz zum Protokoll, welches ausführliche Kenntnisse über die durchgeführten Testabläufe erfordert und primär der Fehleranalyse dient. Somit dient das Test-Reporting als Ergänzung zum Testprotokoll [Repo00]. Ein Testbericht bei Regressionstests sollte folgende Informationen enthalten [HuKo07]:

- Die Summe der ausgeführten Regressionstests sowie die Anzahl gegliedert nach den jeweiligen Ergebnissen
- Ergebnisbeschreibung (= Was hat zu dem jeweiligen Ergebnis geführt?)
- Informationen zur Lokalisierung des Fehlers
- Zusammenfassung der wichtigsten Systeminformationen [Repo00]
- Gab es Testfälle, die nicht funktionierten, weil die Voraussetzungen fehlten oder bei der Testautomatisierung von langen Testabläufen der Weg zum eigentlichen Testobjekt fehlerhaft war und so die Basis fehlte [Witt19]

All diese Informationen sollen dazu dienen, dass Regressionen schnell erkannt, lokalisiert und behoben werden können. Dabei ist darauf zu achten, dass auch Mitarbeiter und Mitarbeiterinnen, welche nicht intensiv mit dem Testobjekt vertraut sind, mit diesen Informationen weitere Maßnahmen ergreifen können. Zudem dienen diese Informationen auch zum Abgleich mit Testergebnissen, welche zu einem späteren oder früheren Zeitpunkt erstellt werden [SpLi19]. Diese historische Aufzeichnung und der Vergleich dieser Informationen ist vor allem beim Regressionstest von entscheidender Bedeutung, da es hier darum geht, herauszufinden, was sich aufgrund von Modifikationen seit dem letzten Testzyklus verändert hat [HuKo07]. Testergebnisse werden für das bessere Verständnis im Testbericht grafisch aufbereitet. Dabei kommen mehrere Darstellungsformen wie Symbole, Diagramme, Farben und Tabellen zur Anwendung [ArOS04].

2.2 Beschreibung Robotic Prozess Automation

RPA ist eine Software, mit der Softwareroboter programmiert werden können, welche strukturierte Aufgaben und Tätigkeiten übernehmen, die bisher von Menschen ausgeführt wurden [GrMT21]. Strukturierte Aufgaben sind Prozesse, welche

Routineaufgaben, semi-strukturierte und strukturierte Daten aufweisen [Bret20]. Diese Softwareroboter interagieren mit dem System über die Benutzeroberfläche gleich, wie es auch der*die Anwender*in macht. Dabei führt der Softwareroboter auch dieselben Prozesse aus. Der Softwareroboter kann im Vergleich zum Menschen schneller, fehlerfreier und durchgehend die Prozesse abarbeiten [LaTu20]. Somit interagiert der Softwareroboter über die Präsentationsebene mit den darunterliegenden Datenbanken und Systemen. Das ist auch der Unterschied zwischen RPA und anderen Automatisierungstools, da andere auf der Ebene der Datenhaltung und Verarbeitung eingreifen [GrMT21].

Zudem sind bei RPA keine besonderen Programmierkenntnisse für die Entwicklung der Softwareroboter erforderlich. Somit können Fachexpertinnen und Fachexperten, welche ausführliche Kenntnisse über die jeweiligen Prozesse besitzen, einen Softwareroboter konfigurieren. Es bedarf lediglich eines Trainings mit der jeweiligen RPA-Software [GrMT21]. Aufgrund der einfachen Entwicklung von solchen Softwarerobotern stellt die IT-Abteilung nur die jeweilige RPA-Software zur Verfügung und legt die technischen Rahmenbedingungen fest. Die Fachabteilung führt dann die Umsetzung der Prozesse mittels RPA durch [ArHH21].

2.2.1 RPA-Architektur

Standardsoftware für RPA kann sich in den Funktionalitäten unterscheiden, aber allen liegt eine grundlegende Architektur zugrunde [CzAu18]. Abbildung 6 zeigt den Aufbau dieser Architektur mit den drei Bereichen Prozess, Ausführung und dem IT-System.

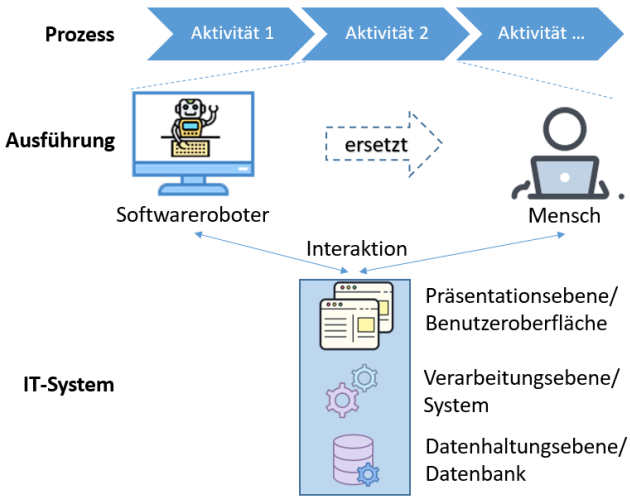


Abbildung 6: Grundlegende Architektur von RPA (eigene Darstellung in Anlehnung an [CzAu18] und [GrMT21])

Ausgangspunkt dieser Architektur sind die Prozesse mit strukturierten Aufgaben, welche zum Teil manuell vom Menschen ausgeführt werden. Dabei sind nicht alle Prozesse oder Teilprozesse für die Umsetzung mittels RPA geeignet. Für die Auswahl geeigneter Prozesse haben sich unterschiedliche Kriterien etabliert. Dazu zählt das Kriterium, dass es sich um regelbasierte Prozesse handeln muss. Das sind Prozesse, bei denen klar ist, welche Aktion bei welchen vorhersehbaren Ereignissen als nächstes durchzuführen ist. Ein weiteres Kriterium ist die Frequenz. Prozesse, die sehr häufig durchlaufen werden, sind besser geeignet als Prozesse, die sehr selten durchlaufen werden [LaTu20]. Das liegt vor allem daran, dass auch die Umsetzung eines Softwareroboters Kosten mit sich bringt, welche durch die Einsparungen der Prozessautomatisierung kompensiert werden müssen [Bret20]. Zudem ist RPA auch gut geeignet für standardisierte Prozesse. Diese sind meist gut dokumentiert, haben keine unerwarteten Ereignisse und folgen einem definierten Ablauf. Das sind sehr stabile und ausgereifte Prozesse im Unternehmen. Zudem muss der Softwareroboter auch mit den im Prozess vorkommenden Daten umgehen können. Daher ist das Vorliegen von elektronischen lesbaren Standarddatentypen ein weiteres Kriterium. Dazu zählen beispielsweise Formate wie CSV, E-Mail oder Webseiten [LaTu20]. Diese Aufzählung an Kriterien ist nicht vollzählig. Die Entscheidung darüber, welche Prozesse und mit welchen Kriterien diese ausgewählt werden, muss am Ende jedes Unternehmen selbst treffen.

Bei der Ausführung der Prozesse mit RPA liegt der Fokus bei einer End-to-End-Betrachtung der Prozesse. RPA imitiert bei der Abarbeitung der Prozesse die menschlichen Arbeitsschritte. Dabei können auch mehrere Systeme eingebunden sein [Bret20]. Softwareroboter führen die Aktivitäten vollautonom und eigenständig aus und kontrollieren am Ende selbst, ob sie den Prozess erfolgreich abgearbeitet haben [LaTu20]. Dabei muss sich auch der Softwareroboter wie der Mensch mit einem Log-in und Passwort am System anmelden und über die Benutzeroberfläche mit dem System interagieren [GrMT21]. Zudem erhält der Softwareroboter eine eigene Nutzerkennung, an welche eigene Rollen und Zugriffsrechte am IT-System verbunden sind [CzAu18].

IT-Systeme lassen sich auf drei Ebenen unterteilen: Die Präsentationsebene, welche durch die Benutzeroberfläche dargestellt wird, die Verarbeitungsebene, das ist der Systemcode, welcher den Input entsprechend verarbeitet und falls erforderlich in der dritten Ebene, der Datenhaltungsebene, ablegt oder weitere Daten abfragt. RPA interagiert mit dem IT-System nur über die Präsentationsebene, alle anderen Ebenen werden nicht direkt angesteuert. Technische Schnittstellen müssen nicht extra für den

Einsatz von RPA eingerichtet werden, somit ist keine Veränderung an der bestehenden IT-Landschaft und an den Systemen bei der Implementierung von RPA erforderlich [ArHH21].

Der Aufbau der RPA-Software selbst lässt sich in drei Komponenten unterteilen. Diese Komponenten sind die Entwicklungskomponente, die Softwareroboter selbst und eine Kontrollkomponente. Bei der Entwicklungskomponente handelt es sich um eine in die RPA-Software integrierte Entwicklungsumgebung für die Softwareroboter. Die einzelnen Aktivitäten des Softwareroboters lassen sich durch vorstrukturiere Aktivitäten und Komponenten anhand von Drag and Drop zusammensetzen. In den einzelnen Aktivitätsfenstern können die jeweils erforderlichen Parameter eingegeben werden. Neben einer grafischen Modellierung der Aktivitäten ist es bei mancher RPA-Software die Modellierung mittels Skripten zu definieren. Die zweite Komponente besteht aus einem oder mehreren Softwarerobotern, welche die vorgegebenen Aktivitäten abarbeiten. Der Einsatz dieser Softwareroboter kann entweder auf einem einzelnen Desktop oder auf einem zentralen Server erfolgen [LaTu20]. Dabei unterscheidet man bei den Softwarerobotern zwei Arten der Steuerung, die „attended“ und die „unattended“ Steuerungsart. Die „attended“ läuft direkt auf dem Desktop und wird von einer Benutzerin oder einem Benutzer gestartet. Die „unattended“ ist jene, die auf einem Server läuft und durch einen entsprechenden Trigger gestartet wird [GrMT21]. In der Kontrollkomponente lassen sich die Softwareroboter zentral ansteuern, überwachen und zeitlich terminieren. Diese Komponente liefert auch Protokolle und Berichte, wo die Zugriffe, die Geschwindigkeit und eventuell auftretende Fehler und Ausnahmen je Software-Bot dargestellt werden [LaTu20]. Vor allem bei der Kontrollkomponente gibt es große Unterschiede in den Ausprägungen der Funktionen. Einige RPA-Tools bringen alle beschriebenen Funktionen mit, andere nur jene der zentralen Ansteuerung.

2.2.2 Funktionsweise

Damit der Softwareroboter die Arbeit aufnehmen kann, braucht es einen Trigger. So ein Trigger kann ein eintretendes Ereignis wie der Eingang einer E-Mail sein oder der Einsatz des Softwareroboters ist zeitlich terminiert und er wird immer zu genau festgelegten Zeiten gestartet [Bret20]. Die zeitliche Steuerung wird vor allem dazu genutzt, um Prozesse in der Nacht abzuarbeiten, damit der Betrieb am Tag nicht eingeschränkt wird [ArHH21]. Zudem können Softwareroboter auch manuell zu einem beliebigen Zeitpunkt durch den Menschen gestartet werden. Ist der Softwareroboter

gestartet, arbeitet er die ihm vorgegebenen Aktivitäten ab. Abbildung 7 zeigt eine vereinfachte Darstellung über die automatisierte Bedienung einer Benutzeroberfläche.

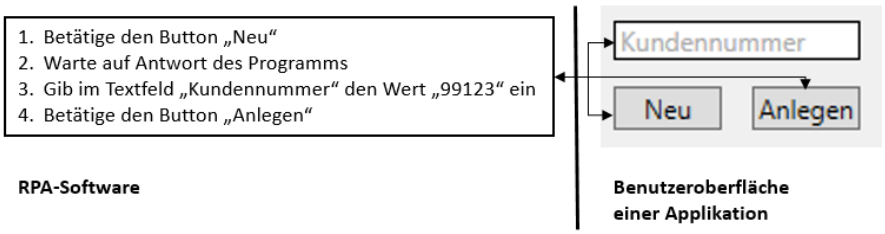


Abbildung 7: Bedienung einer Benutzeroberfläche mittels RPA (eigene Darstellung in Anlehnung an [Bret20])

Auf der linken Seite der Abbildung 7 sind die einzelnen Schritte bzw. Steuerbefehle für den Softwareroboter dargestellt. Auf der rechten Seite ist eine Benutzeroberfläche einer Applikation abgebildet, welche aus einigen Steuerelementen besteht. Der Softwareroboter geht die Schritte einzeln durch und bedient dabei die Benutzeroberfläche. Die Erkennung und die Ansteuerung der einzelnen Elemente in der Benutzeroberfläche können auf unterschiedliche Arten erfolgen. Entweder über die im Hintergrund vorhandenen Eigenschaftswerte eines Steuerelementes wie zum Beispiel die Automation ID, über eine Tastenkombination oder über die optische Erkennung des Steuerelementes [Uipa00a]. Die Aktion, die auf das Steuerelement ausgeführt werden kann, umfasst alle Aktivitäten, die der Mensch auch durchführen kann. Sind alle Prozesse abgearbeitet oder ist innerhalb der Prozesse ein unerwartetes Ereignis eingetreten und der Softwareroboter konnte den Prozess nicht erfolgreich abschließen, kann am Ende eine Information zum Beispiel mittels E-Mail an einen bestimmten Personenkreis erfolgen [Bret20].

2.2.3 Anwendungsszenarien

In vielen Unternehmen ist die IT-Landschaft über die Jahre gewachsen und unterschiedliche, zum Teil nicht miteinander kompatible Systeme, haben sich gebildet, wodurch die Komplexität der IT-Landschaft zugenommen hat. Die Umstellung – weg von bereits etablierten IT-Systemen zu einem neuen einheitlichen IT-System wie ein ERP-System – ist meist kosten- und zeitintensiv. Vielmehr wird auf zusätzliche Mitarbeiter und Mitarbeiterinnen gesetzt, welche diese Schnittstellenprobleme in Form von Routineaufgaben übernehmen [BFGL18]. Genau dort liegt das Einsatzgebiet für RPA - bei IT-Systemen, wo Prozesse systemübergreifend vorhanden sind, aber Schnittstellen fehlen, und wo die Mitarbeiter und Mitarbeiterinnen von einfachen monotonen Tätigkeiten befreit werden können [GrMT21]. Die Einsatzmöglichkeit von

RPA reicht von einfachen bis sehr komplexen Prozessen. Dabei wird bei den Prozessen zwischen folgenden Komplexitätsgraden unterschieden [CzBA19]:

1. *Routineaufgaben*, wo aus unterschiedlichen Anwendungssystemen die Daten kopiert und kombiniert werden.
2. *Strukturierte Aufgaben*, wo Daten aus unterschiedlichen Anwendungssystemen genutzt und mithilfe eines definierten Regelwerkes bewertet werden.
3. *Unstrukturierte Aufgaben und Entscheidungen*, wo entsprechendes Entscheidungswissen neben den Daten und Regelwerken erforderlich ist.

Zu den Routineaufgaben zählen Prozesse, welche repetitive, strukturierte Tätigkeiten umfassen und wo die Interaktion mit verschiedenen nicht integrierten Anwendungssystemen erforderlich ist [CzBA19]. Typische Aufgaben, die mit RPA in diesem Bereich übernommen, werden sind [KoFe20]:

- Befüllen von Formularen
- Einfügen und kopieren von Datensätzen
- Verschieben von Dateien und Ordnern
- Ausführen von Kalkulationen

Strukturierte Aufgaben stellen eine Erweiterung der Routineaufgaben dar, indem regelbasierte Entscheidungen in den Prozess mitaufgenommen werden müssen. Gibt es strukturierte Prozesse, wo Entscheidungen durch klare Geschäftsregeln beschrieben sind, können diese auch durch RPA automatisiert werden. Typische Aktivitäten von RPA bei strukturierten Aufgaben sind [CzBA19]:

- Kombinieren und bewerten von Daten aus unterschiedlichen Quellen
- Erstellen von Berichten
- Verarbeiten von E-Mails
- Regelbasiertes Ausführen von Funktionen in den Anwendungssystemen

Bei den unstrukturierten Aufgaben erfolgt die Kombination von RPA mit anderen Systemen wie Natural Language Processing, Deep Learning oder Machine Learning [ArHH21]. Darunter fallen Prozesse, welche unstrukturierte Aufgaben und Entscheidungen enthalten und die Erfahrung aus der Vergangenheit oder Entscheidungswissen erfordern [CzBA19]. Typische Anwendungsfälle sind:

- Strukturieren von unstrukturierten Daten [ArHH21]
- Erkennen von Risiken [CzBA19]

2.2.4 RPA-Tools

In diesem Abschnitt werden vier RPA-Tools verglichen. Dabei handelt es sich um zwei Lizenzprodukte und zwei Open-Source-Lösungen. Für die Lizenzprodukte wurden UiPath und Automation Anywhere gewählt. Diese Anbieter sind seit Jahren Marktführer im Bereich RPA-Software. Als Open-Source-Lösungen wurden OpenRPA und taskt ausgewählt. Beide Open-Source-Lösungen liefern ein breites Angebot an Funktionen und beinhalten eine sehr gute Dokumentation, sodass die Einarbeitungsphase gut unterstützt wird. In Tabelle 2 sind die Unterschiede und Gemeinsamkeiten dieser RPA-Tools zusammengefasst. Dabei wurde speziell auf jene Bereiche eingegangen, welche für den Einsatz von RPA für die Testautomatisierung von Relevanz sind.

RPA-Anbieter	UiPath	Automation Anywhere	OpenRPA	taskt
Versionsnummer	2022.4.4	11.3	1.4.35	3.5.0.0
Entwicklung	Drag-and-Drop-Funktionen Workflow-orientierte Entwicklung	Drag-and-Drop-Funktionen Skript- und Workflow-orientierte Entwicklung	Drag-and-Drop-Funktionen Workflow-orientierte Entwicklung	Drag-and-Drop-Funktionen Skript-orientierte Entwicklung
Erweiterungen	DLL	DLL	DLL	DLL
Start von Command Line	Ja	Ja	Ja	Ja
Recording	Ja	Ja	Ja	Ja
Desktop Automation	Ja	Ja	Ja	Ja
Kontrollkomponente	Ja	Ja	Keine, nur eine Möglichkeit für die Auswahl der Softwareroboter	Keine
Lizenzbedingungen	60-Tage-Testversion, mehrere Lizenzmodelle je nach Bedarf	30-Tage-Testversion, mehrere Lizenzmodelle je nach Bedarf	Kostenlos, Open Source	Kostenlos, Open Source

Tabelle 2: Vergleich der RPA-Tools (eigene Zusammenstellung aus den Quellen [Uipa00b], [Auto00], [VPTB00], [Task22])

In der Übersicht sind zu Beginn die jeweiligen RPA-Anbieter und die jeweiligen Versionsnummern, auf die sich diese Angaben beziehen, angeführt. Bei der Entwicklung unterscheiden sich die RPA-Tools in der Darstellung des jeweiligen Prozesses. Da gibt es die Variante der Workflow-orientierten Entwicklung, wo mit kleinen Symbolen und Grafiken der Workflow dargestellt wird oder die Skript-orientierte Entwicklung, bei der die einzelnen Schritte untereinander als Skript bzw. Textteile aufgelistet werden. Abbildung 8 zeigt einen Vergleich zwischen Workflow-orientierte Entwicklung und Skript-orientierte Entwicklung. Auf der linken Seite ist ein Beispiel einer Workflow-orientierten Entwicklung, wie sie bei UiPath und OpenRPA zum Einsatz kommt und auf der rechten Seite ist ein Beispiel einer Skript-orientierten Entwicklung, wie es taskt anbietet.

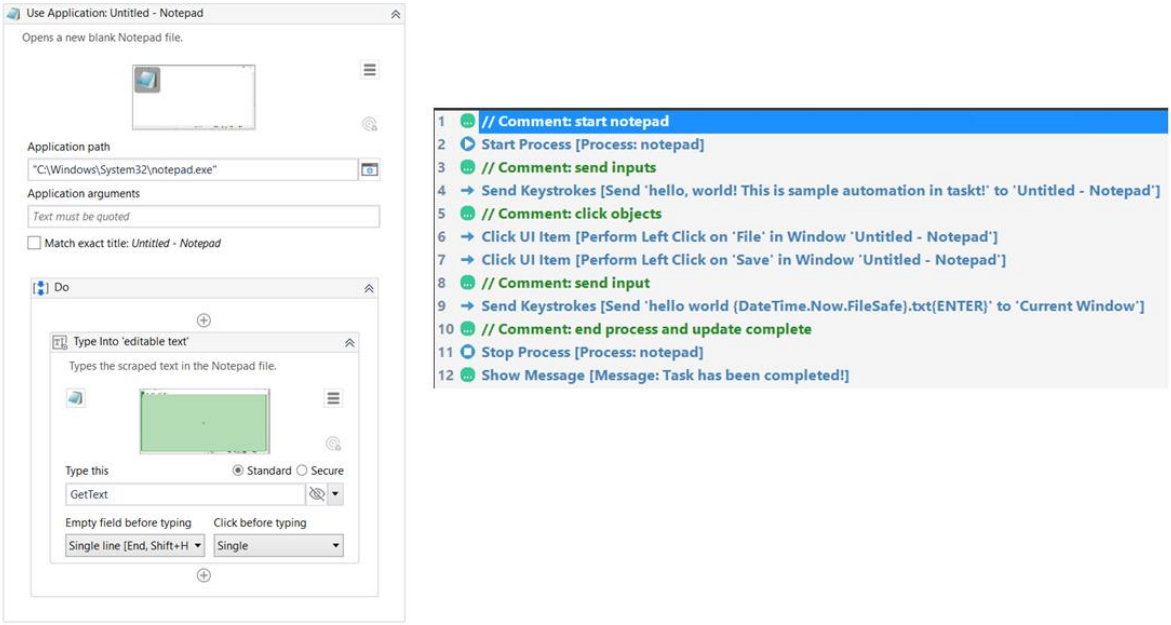


Abbildung 8: Vergleich Entwicklungsarten [Uipa00c],[Task22]

Beide Entwicklungsarten werden dabei durch einfache Drag-and-Drop-Funktionen unterstützt, wo bereits vordefinierte Aktivitäten bzw. Funktionsbausteine einfach zu einem Prozess zusammengesetzt werden können.

Gibt es keine passende Aktivität, können die RPA-Tools durch eine Dynamic Link Library (DLL) erweitert werden. Diese DLLs sind Dateien, welche Daten und Code enthalten. Sie finden im .Net Framework Einsatz und dienen dazu, ein Programm in einzelne Module aufzuteilen. Zudem können mehrere Programme gleichzeitig diese DLLs verwenden [Dela00]. In unserem Anwendungsfall können wir diese DLLs dazu verwenden, neue eigene Funktionen in den RPA-Tools hinzuzufügen. Hierfür sind aber Programmierkenntnisse erforderlich, da diese DLLs meist in C# oder C++ geschrieben werden. Da alle RPA-Tools über diese Möglichkeit verfügen, sind die DLLs eine geeignete Variante für die Einbindung von RPA in das Framework.

Bei allen RPA-Tools ist es möglich, die Softwareroboter über die Command Line zu starten. Dazu sind die Angabe des Pfades zum Installationsverzeichnis der jeweiligen RPA-Software und ein entsprechender Parameter für den jeweiligen Softwareroboter erforderlich. Je nach RPA-Tool ist dieser Parameter der Name des Prozesses, der hinter dem Softwareroboter steckt, oder eine ID. Auch diese Möglichkeit ist wichtig, damit die Softwareroboter beim Einbinden in das Framework einzeln gesteuert werden können.

Alle RPA-Tools besitzen eine Recording-Funktion. Mit dieser Funktion können schnell und einfach Prozesse aufgezeichnet werden. Der*die Anwender*in führt die

entsprechenden Schritte aus und das RPA-Tool modelliert die Prozesse selbst. Diese können danach sofort oder nach kleineren Anpassungen durch den Softwareroboter wiederholt werden.

Für die Testautomatisierung mit RPA ist es wichtig, dass die eingesetzten RPA-Tools für die Desktopautomatisierung geeignet sind. Die jeweiligen Testfälle werden über den*die Anwender*in gestartet und müssen auf einem Desktoprechner in einer realen oder virtuellen Umgebung ausgeführt werden. Alle hier beschriebenen RPA-Tools besitzen die Möglichkeit zur Desktopautomatisierung.

Bei der Kontrollkomponente gibt es vor allem bei den Open Source RPA Tools große Unterschiede. UiPath und Automation Anywhere besitzen eine Kontrollkomponente, mit der die Steuerung und auch die Überwachung möglich sind. OpenRPA hat nur die Möglichkeit der Steuerung und taskt besitzt keine Kontrollkomponente. Bei taskt müssen die einzelnen Softwareroboter über die Entwicklungsumgebung gestartet werden [Task22].

OpenRPA und taskt sind kostenlose Open-Source-Anwendungen. Diese können über GitHub heruntergeladen und installiert werden. UiPath bietet eine 60-tägige Testversion an und danach muss man sich für ein entsprechendes Lizenzmodell entscheiden. Je nach Bedarf werden einem mehrere Modelle angeboten [Uipa00b]. Dasselbe gilt auch für Automation Anywhere, aber hier gibt es nur eine 30-tägige Testversion [Auto00].

2.3 Zusammenfassung

Der Testprozess dient vor allem als organisatorischer Rahmen für die Einbindung von RPA in die Testautomatisierung. Bereits in der Testplanung muss entschieden werden, was automatisiert wird und ob es mit RPA automatisiert werden soll. Danach erfolgt die Testrealisierung und Testdurchführung bereits mit der Automatisierung durch RPA. Dieser Testprozess hat auch gezeigt, dass es wichtig ist, am Ende einen Bericht über die Ergebnisse zu haben, der evaluiert und für spätere Tests archiviert wird. Bei den unterschiedlichen Teststufen kann RPA bei den Systemtests und Abnahmetests eingesetzt werden. Softwareroboter interagieren über die Benutzeroberfläche mit der Applikation. So können diese ganze Geschäftsprozesse, oder Teile davon, auf dem gesamten System abarbeiten. Der prinzipielle Aufbau einer Framework-Architektur zeigt, wie RPA in diese eingebettet werden kann, daraus können auch wichtige Anforderungen für das Framework abgeleitet werden. Regressionstests sind

wiederkehrende gleichbleibende Tests, welche negative Auswirkungen von Modifikationen auf bestehende Funktionen erkennen sollen. Da hat sich gezeigt, dass es hier mehrere Ansätze gibt, diese durchzuführen und dass hier die Steuerung der Testfälle eine wesentliche Rolle spielt. Die Entwicklung und Durchführung solcher Regressionstests kosten Zeit und Geld und müssen daher sorgfältig geplant werden. Die Beschreibung von End-to-End-Tests zeigt die Gemeinsamkeiten von dieser Testart und der Funktionsweise bzw. der Anwendungsszenarien von RPA. In beiden Bereichen wird über die Benutzeroberfläche aus Sicht des Anwendenden mit dem System interagiert. Am Ende jedes Tests braucht es ein gut strukturiertes, übersichtliches und leicht verständliches Test-Reporting. Es muss so gestaltet sein, dass sich auch jene Personen, welche sich nicht mit den einzelnen Testfällen im Detail auskennen, mit den Ergebnissen etwas anfangen können. Zudem sollte das Test-Reporting bei Regressionstests so gestaltet sein, dass es mit historischen Ergebnissen verglichen werden kann. Die grundlegende Architektur von RPA gibt Aufschluss darüber, wie Softwareroboter eingesetzt werden und aus welchen Komponenten RPA-Tools bestehen. Dabei konnten Ähnlichkeiten zwischen dem Einsatz von RPA und von End-to-End-Tests festgestellt werden. Die Komponenten von RPA-Tools umfassen alles, von der Entwicklung bis zur Steuerung der Softwareroboter. Dabei ist die Entwicklung der Softwareroboter so gestaltet, dass diese ohne Programmierkenntnisse erstellt werden können. Bei der Funktionsweise wurde dargestellt, wie die Prozesse in der RPA-Software abgebildet werden und wie der Softwareroboter mit der Benutzeroberfläche interagiert. Am Ende wurden vier verschiedene RPA-Tools verglichen, um Unterschiede und Gemeinsamkeiten dieser Tools herauszufinden. Verglichen wurden zwei Lizenzprodukte und zwei Open-Source-Lösungen. Dabei wurde vor allem nach Gemeinsamkeiten gesucht, welche für die Entwicklung eines Frameworks für die Testautomatisierung mittels RPA relevant sind. Es wurde festgestellt, dass alle hier verglichenen RPA-Tools Funktionen besitzen, welche die Einbindung in ein solches Framework ermöglichen und diese RPA-Tools durch ihre Möglichkeit der Desktop Automatisierung auch für die Testautomatisierung geeignet sind. Bei der Ausarbeitung der Grundlagen und dem aktuellen Stand der Technik konnten viele Eigenschaften aus den Bereichen der Testautomatisierung und RPA herausgearbeitet werden, welche für das nächste Kapitel, der Konzeption des Frameworks, vor allem für die Anforderungsanalyse wichtig sind.

3 Konzeption eines Frameworks für die Testautomatisierung mit RPA

Im folgenden Kapitel wird das Framework für die Testautomatisierung mit RPA dargelegt. Zu Beginn werden die Ziele des Frameworks vorgestellt und die Anforderungen aufgelistet und beschrieben. Auf Basis der Ziele erfolgt die Konzeption des Frameworks. Die Beschreibung des Konzepts beginnt mit der Darstellung der gesamten Architektur des Frameworks und mit der detaillierten Beschreibung der Einbindung von RPA-Tools in das Framework. Am Ende werden die wesentlichen Prozesse aus Sicht der*des Anwendenden beschrieben und grafisch dargestellt.

3.1 Ziele des Frameworks

Ein grundsätzliches Ziel des Frameworks ist, damit eine einfache Steuerung der Softwareroboter für die einzelnen Testfälle zu ermöglichen. Dabei sollen dieselben Entwicklungsansätze für den*die Anwender*in zum Einsatz kommen wie bei RPA. Das bedeutet, es sollen keine Programmierkenntnisse für die Einbindung der Softwareroboter von dem*der Anwender*in erforderlich sein. Ein weiteres Ziel ist die systematische Protokollierung der Fortschritte und ein grafisch aufbereitetes Test-Reporting. Zwar soll der Softwareroboter auch bei der Testautomatisierung bei einfachen Prozessen zum Einsatz kommen, da es sich hier aber um einen Test des Systems handelt und dabei Fehler auftreten können, muss für eine Rückverfolgung der Fehler der jeweilige Fortschritt protokolliert werden.

Das Framework ist als prototypisches Framework zu betrachten bzw. soll es auch eine Referenzarchitektur darstellen, bei der die Softwareroboter implementiert werden können. Die bei der RPA fehlenden Bestandteile wie die Steuerung, Protokollierung und das Test-Reporting sollen mit dem Framework bei den eingesetzten RPA-Tools durch eine prototypische Umsetzung des Frameworks ergänzt werden.

Es ist nicht das Ziel, eine voll funktionsfähige Software zu entwickeln. Das Schwergewicht der Konzeption und der Entwicklung des Frameworks liegt in jenen Bereichen, welche für die Beantwortung der Forschungsfrage von Bedeutung sind. Das betrifft somit eine Bereitstellung der Komponenten, welche für die Steuerung, dem Test-Reporting sowie für eine prototypische Implementierung erforderlich sind.

3.2 Anforderungen an das Framework

Im folgenden Abschnitt werden die Anforderungen an das Framework beschrieben. Diese bilden die Basis für die Konzeption des Frameworks und für die prototypische Implementierung. Wie bereits bei den Zielen erwähnt, beschränken sich auch die Anforderungen auf die im Zusammenhang mit der Forschungsfrage stehenden Bereiche. Die Anforderungen werden mit der Abkürzung „Anf. [fortlaufende Nummer]“ bezeichnet, damit später auf die einzelnen Anforderungen referenziert werden kann.

Die Anforderungen sind für einen besseren Überblick in die drei Bereiche grundlegende Anforderungen, Anforderungen an die Steuerung der Softwareroboter und Anforderungen an das Test-Reporting unterteilt. Die nachfolgenden Anforderungen werden aus den Zielen für das Framework sowie aus den im 2. Kapitel beschriebenen allgemeinen Funktionsweisen und Prozessen bei der Testautomatisierung und beim Einsatz von RPA abgeleitet.

3.2.1 Grundlegende Anforderungen an das Framework

In diesem Abschnitt werden die grundlegenden Anforderungen aufgelistet, welche für das gesamte Framework gelten.

Anf. 1: Die Architektur des Frameworks soll sich an der im Abschnitt 2.1.3 beschriebenen Framework-Architektur für die Testautomatisierung orientieren.

Anf. 2: Die Implementierung der Softwareroboter in das Framework soll ohne Programmierkenntnisse der*des Anwendenden möglich sein.

Anf. 3: Die Softwareroboter sollen individuell von dem*der Anwender*in über die von den RPA-Tools zur Verfügung gestellten Entwicklungskomponenten entwickelt und angepasst werden können.

Anf. 4: Das Framework muss so gestaltet sein, dass es zumindest mit den im Abschnitt vorgestellten RPA-Tools für die Testautomatisierung eingesetzt werden kann.

Anf. 5: Mit dem Framework muss eine Realisierung der Testfälle anhand des im Abschnitt 2.1.1 vorgestellten Testprozesses möglich sein.

Anf. 6: Das Framework muss zu jedem Testfall die Testdaten sowie weitere für den Testfall erforderliche Daten oder Dateien strukturiert verwalten können.

Anf. 7: Die Testdaten müssen auch aus dem Framework heraus bearbeitet werden können.

Anf. 8: Die einzelnen Testfälle müssen beschrieben und dem*der Anwender*in angezeigt werden können.

3.2.2 Anforderung an die Steuerung der Softwareroboter

Dieser Abschnitt beschreibt spezielle Anforderungen, welche nur die Steuerung der Softwareroboter mit dem Framework betreffen.

Anf. 9: Das Framework soll den Einsatz von „attended“ RPA unterstützen und mehrere Testfälle sequenziell abarbeiten können.

Anf. 10: Das Starten der Softwareroboter muss mittels Eingabe eines Skriptpfades oder der Skript ID möglich sein.

Anf. 11: Es muss möglich sein, dass ein oder mehrere Testfälle von dem*der Anwender*in ausgewählt und gestartet werden können.

Anf. 12: Die Reihenfolge wie die Testfälle bei einem Testdurchlauf abgearbeitet werden, muss von dem*der Anwender*in veränderbar sein.

3.2.3 Anforderung an das Test-Reporting

In diesem Abschnitt werden spezielle Anforderungen an das Test-Reporting hinsichtlich Funktionalität und Design beschrieben.

Anf. 13: Die strukturierte Protokollierung von benutzerdefinierten Fortschritten beim Durchlaufen der Prozesse muss möglich sein.

Anf. 14: Nach der Ausführung der Testfälle braucht es am Ende eine gesamte Übersicht der durchgeführten Testfälle sowie der Dauer und der Ergebnisse.

Anf. 15: Für jeden Testfall soll es am Ende auch einen Detailbericht über die Fortschritte, die Ergebnisse und die Dauer geben.

Anf. 16: Für ein leichtes Verständnis und einen guten Überblick soll das Test-Reporting zum Teil auch grafisch aufbereitet sein.

Anf. 17: Die Daten für das Test-Reporting müssen archiviert werden können.

Anf. 18: Testergebnisse aus vergangenen Tests müssen bei einem erneuten Durchführen der Testfälle in das Reporting einbezogen und verglichen werden können.

Anf. 19: Das Test-Reporting soll den im Abschnitt 2.1.6 beschriebenen Erfordernissen entsprechen.

Anf. 20: Das Hinzufügen von Anmerkungen zum Testdurchlauf muss möglich sein.

3.3 Gesamtarchitektur des Frameworks

Im folgenden Abschnitt wird die Gesamtarchitektur des Frameworks vorgestellt. Diese Architektur wurde aus den Anforderungen abgeleitet. Zudem wurde darauf geachtet, wie RPA in die Framework-Architektur für die Testautomatisierung integriert werden kann. Abbildung 9 zeigt eine schematische Darstellung der Gesamtarchitektur des Frameworks.

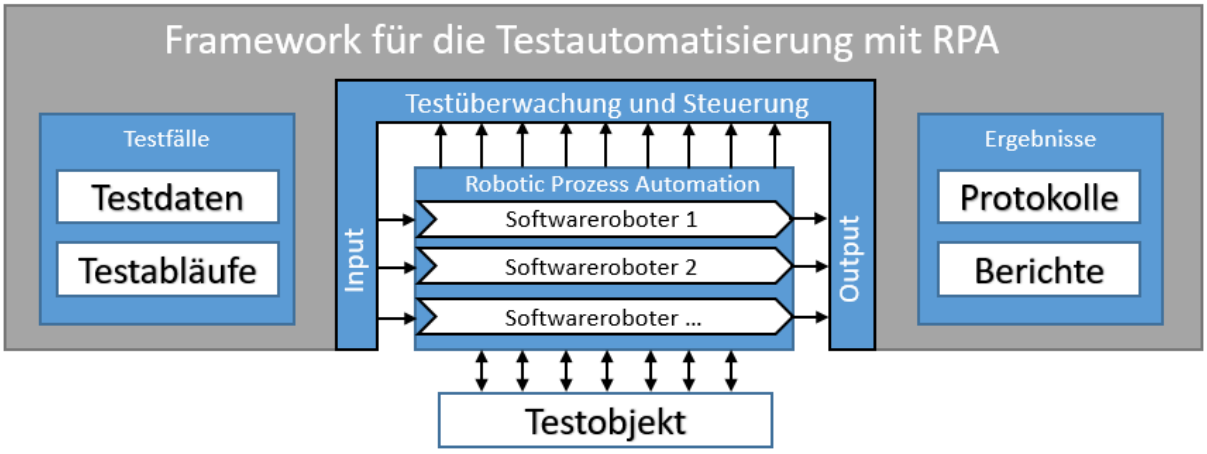


Abbildung 9: Architektur des Frameworks für die Testautomatisierung mit RPA (eigene Darstellung)

Wie man aus der oben angeführten Darstellung erkennen kann, gibt es viele Gemeinsamkeiten mit der Framework-Architektur für die Testautomatisierung [Anf. 1]. Es muss bei diesem Framework auch entsprechende Testfälle in Form von Testdaten und Testabläufen geben und am Ende braucht es ein entsprechendes Ergebnis in Form von Protokollen und Berichten. Der wesentliche Unterschied liegt im Zentrum des Frameworks. Hier ist die RPA-Software eingebettet und es bedarf hier mehrerer Schnittstellen für die Steuerung, die Überwachung und für den Output der Ergebnisse. Zudem erfolgt der Zugriff auf das Testobjekt ausschließlich über die Softwareroboter.

Die Softwareroboter arbeiten die entsprechenden Testfälle sequenziell ab und liefern dabei entsprechende Fortschrittsinformationen an das Framework.

Mit diesem Framework kann die Verwirklichung der Testautomatisierung anhand des im Abschnitt 2.1.1 beschriebenen Testprozesses durchgeführt werden [Anf. 5]. Bei der Testanalyse und beim Testentwurf muss nur die technische Machbarkeit mit der jeweiligen RPA-Software geprüft werden und ob die Umsetzung in einem vertretbaren Aufwand gemacht werden kann. Die Testrealisierung erfolgt in zwei Schritten. Zu Beginn wird der Prozess für den Testfall mit den Softwarerobotern in der jeweiligen Entwicklungskomponente der RPA-Software umgesetzt. Im zweiten Schritt werden die fertigen Softwareroboter in das Framework integriert. Für die Testdurchführung können die Testfälle aus dem Framework heraus gestartet werden und am Ende wird für die Bewertung der Testfälle ein entsprechender Bericht erstellt.

3.3.1 Einbindung der Testfälle in das Framework

Die Einbindung der Testfälle erfolgt dadurch, dass die Testdaten strukturiert abgelegt werden, die Beschreibung der Testabläufe im Framework angezeigt wird sowie durch eine Auflistung der vorhandenen Testfälle zur Auswahl für die Anwendenden. Die Testdaten werden je Testfall in einen eigenen Ordner abgelegt. In diesem Ordner befindet sich eine Excel-Datei, in der für den Testfall relevante Daten enthalten sind, und weitere individuelle Dateien, sofern diese für den Testfall notwendig sind. In der Excel-Datei sind die Informationen zum Soll-Ist-Vergleich der Ergebnisse, es können aber auch Steuerinformationen für den Softwareroboter darin enthalten sein. Auf die Excel-Datei können alle RPA-Tools einfach zugreifen und die benötigten Daten auslesen. Zudem können diese Excel-Dateien aus dem Framework heraus gestartet werden, was einen schnellen Zugriff ermöglicht [Anf. 7].

Zu jedem Testfall muss es eine Beschreibung geben, in welche der*die Anwender*in den Testfall nach eigenen Bedürfnissen beschreiben kann. Diese Beschreibung soll dem*der Anwender*in bei der Auswahl der Testfälle angezeigt werden [Anf. 8].

Alle Testfälle können nach Bereichen gegliedert werden. Im Framework besteht jeder Testfall aus einer Extensible Markup Language (XML) Datei, den Testdaten, einer RTF-Datei für die Beschreibung und, sofern erforderlich, auch dem jeweiligen Skript für den Softwareroboter. Das ist aber optional und der*die Anwender*in kann selbst entscheiden, da Skripte für den Softwareroboter auch zentral von der RPA-Software

verwaltet werden können. In der XML sind folgende Informationen für den Testfall vorhanden:

- Eine eindeutige Bezeichnung des Testfalles
- Bereich
- Bezeichnung Beschreibungsdatei
- ID des Skripts für den Softwareroboter oder
- Pfad zum Skript für den Softwareroboter [Anf. 10]

Dadurch, dass die Informationen zum Testfall als XML gespeichert werden, ist es möglich, dass das Framework die für die Anzeige erforderlichen Informationen bereitstellen kann und die entsprechenden Skriptinformationen für die Steuerung der Softwareroboter zur Verfügung hat. Im Anhang A ist ein Beispiel für eine solche XML angeführt. Der*die Anwender*in kann über ein Formular in der Benutzeroberfläche die entsprechenden Informationen zu den Testfällen eingeben und diese speichern [Anf. 10]. Das Framework erstellt die XML mit den eingegebenen Daten bzw. ändert diese, wenn die XML schon vorhanden ist. Durch die Eingabe der Informationen in das Formular und durch das Anlegen der entsprechenden Testdaten ist die Implementierung des jeweiligen Softwareroboters abgeschlossen. Es bedarf somit für das Implementieren keine Programmierkenntnisse des Anwendenden [Anf. 2]. Abbildung 10 zeigt die Ablagestruktur der Testfälle im Framework [Anf. 6]:

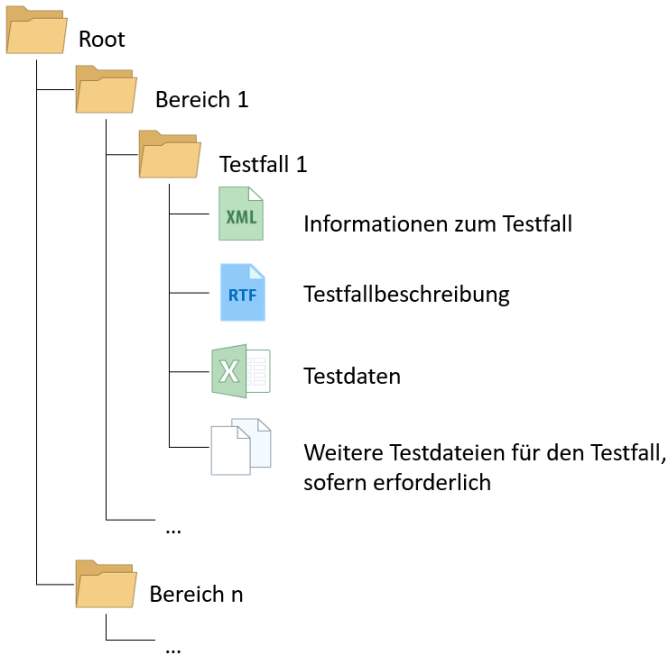


Abbildung 10: Ablagestruktur der Testfälle im Framework (eigene Darstellung)

3.3.2 Einbindung von RPA in das Framework

Jeder Softwareroboter kann mit der von der RPA-Software zur Verfügung gestellten Entwicklungskomponente erstellt werden. Dieser Prozess ist unabhängig vom Framework [Anf. 3]. Der fertige Softwareroboter wird, wie im letzten Abschnitt beschrieben, dem Framework hinzugefügt. Auch die Ablage der Skripte kann individuell erfolgen. Ausschließlich die Softwareroboter interagieren mit dem Testobjekt.

Der Start der Softwareroboter erfolgt über den Start der entsprechenden RPA-Software und mit der Übergabe der ID oder der Bezeichnung bzw. dem Pfad des Skripts für den Softwareroboter. Abbildung 11 zeigt den Start von OpenRPA mit einem Softwareroboter.

```
Start-Process -FilePath "C:\\Program Files\\OpenRPA\\OpenRPA.exe"  -ArgumentList "-workflowid 5f8dfaf243e8cf24d4c25dcd"
```

Abbildung 11: Start von einem OpenRPA Softwareroboter aus der Command Line [VPTB00]

So lassen sich aus dem Framework heraus alle Softwareroboter einzeln starten. Bei jedem Start wird gewartet, ob der Prozess beendet wurde und danach startet der nächste. Durch diese Methode der Steuerung werden die Softwareroboter nach der Methode „attended“ eingesetzt und so wird auch eine sequenzielle Abarbeitung der Testfälle verwirklicht [Anf. 9].

In die RPA-Software werden durch eine DLL zusätzliche Funktionen geschaffen. Diese ermöglicht es, dass der Softwareroboter Fortschrittsinformationen und Ergebnisse speichern kann. Diese Fortschrittsinformationen dienen am Ende als Protokolle. Damit kann geprüft werden, ob der Prozess vollständig durchlaufen wurde oder ob es beim Durchlaufen des Prozesses zu einem Fehler gekommen ist. Diese Fortschrittsinformationen können im Rahmen der Entwicklung des Softwareroboters durch den*die Anwender*in individuell gesetzt werden. Zudem werden in diese Protokolle auch die Ist-Ergebnisse hinzugefügt sowie am Ende das Ergebnis des Soll-Ist-Vergleiches eingetragen. Dieses Protokoll wird in Form einer XML erstellt. Beim Start des jeweiligen Softwareroboters werden bereits folgende Informationen hinzugefügt:

- Bezeichnung des Testfalles
- Bereich
- Timestamp für den Start

Während des Testdurchlaufes werden durch den Softwareroboter folgende Informationen protokolliert:

- Fortschrittshinweis mit Timestamp
- Die Soll- und die Ist-Werte mit Timestamp
- Ergebnis des Soll-Ist-Vergleiches mit Timestamp

Am Ende des Testdurchlaufes werden durch das Framework folgende Informationen ergänzt:

- Je Ergebnis wird der Soll- und der Ist-Wert verglichen und das Ergebnis hinzugefügt
- Timestamp für das Ende
- Die Dauer wird ermittelt und hinzugefügt
- Das Gesamtergebnis für den Testfall wird ermittelt und ergänzt
- Wenn vorhanden, wird das historische Ergebnis hinzugefügt

Das Gesamtergebnis ergibt sich aus dem schlechtesten Ergebnis der einzelnen Ergebnisse. Error ist schlechter als Fail und Fail ist schlechter als OK. Error wird entweder durch den Softwareroboter bei einem Ausnahmefehler eingetragen oder wenn der Testfall nicht durch den Softwareroboter beendet werden konnte, dann wird durch das Framework Error eingetragen.

Diese Protokollierung findet für jeden Softwareroboter in einer eigenen Datei statt [Anf. 13]. Im Anhang B ist ein entsprechendes Muster dieser XML angeführt.

Das Framework startet den Softwareroboter und wartet, bis dieser den Testfall abgearbeitet hat. Danach startet er den nächsten Softwareroboter. Wenn alle durch den*die Anwender*in ausgewählten Testfällen abgearbeitet sind, werden folgende Informationen aus jedem der Protokolle ausgelesen und in eine weitere XML für den Bericht geschrieben:

- Anzahl der Testfälle, gegliedert nach dem Gesamtergebnis
- Start, Ende und Dauer des gesamten Testdurchlaufes
- Systeminformationen
- Anmerkungen zum Testdurchlauf
- Je Testfall
    - Bezeichnung

- Gesamtergebnis
- Historisches Ergebnis
- Zeitpunkt des historischen Ergebnisses

Aus dieser XML und aus den Protokollen wird danach der Bericht erstellt.

Die Voraussetzungen für das Einsetzen einer RPA-Software mit dem Framework zur Testautomatisierung sind, dass einerseits der Start über die Command Line möglich ist und andererseits müssen die Funktionen der RPA-Software über DLL erweiterbar sein [Anf. 4].

3.3.3 Erstellung des Test-Reportings

Die erstellten XML-Dateien werden mit Extensible Stylesheet Language Transformations (XSLT) in Hypertext Markup Language (HTML) Dateien transformiert. Mit XSLT-Stylesheets können XML-Dateien transformiert werden. Den Input bilden dabei immer XML-Dateien. Der Output kann eine XML-, Text- oder wie in unserem Fall eine HTML-Datei sein [Lenz06]. Für die Transformation wird ein XSLT-Prozessor benötigt. Der bei diesem Framework verwendete Prozessor ist Microsoft XML Core Services (MSXML) [Msxm16]. Dieser wird von Microsoft für die Verarbeitung von XML-Dateien zur Verfügung gestellt und ermöglicht die Verwendung von XSLT 1.0. Durch den Einsatz von XSLT für die Datenaufbereitung ist ein individuelles Anpassen der Darstellung jederzeit möglich, ohne den Sourcecode des Frameworks verändern zu müssen, da es sich dabei um eigene Dateien handelt, die jederzeit angepasst werden können. Abbildung 12 zeigt den Transformationsprozess von XML-Dateien in HTML-Dateien mit XSLT-Stylesheets.

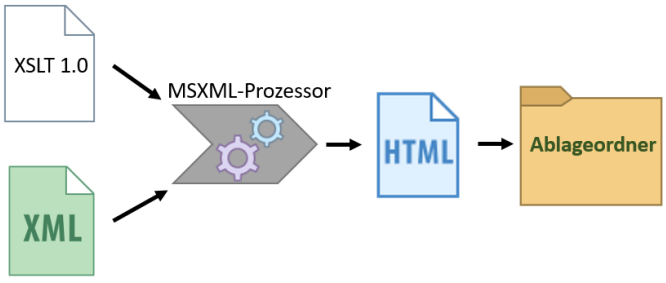


Abbildung 12: Transformationsprozess von XML zu HTML (eigene Darstellung in Anlehnung an [Holm00])

Die so erstellten Dateien werden für jeden Testdurchlauf in einen eigenen Ordner abgelegt. Über das Framework können die HTML-Dateien geöffnet und angezeigt werden. Der Bericht ist der Einstiegspunkt für die Übersicht der Ergebnisse und der

Dauer der Testabläufe [Anf. 14]. Aus dem Bericht heraus kann weiternavigiert werden zu den einzelnen Protokollen, wo weitere Details für die Testabläufe angeführt sind [Anf. 15]. Durch die Verwendung von HTML für die Darstellung der Ergebnisse ist eine einfache und schnelle Aufbereitung der Daten möglich, zudem können entsprechende JavaScript Libraries für die grafische Aufbereitung eingebunden werden. Für den Bericht wird die Library Chart.js verwendet. Damit ist es möglich, einfache Diagramme mit JavaScript zu erstellen [Char00]. Im Bericht wird mit Diagrammen eine Übersicht über die Ergebnisse erstellt. Dabei wird mit einem Balkendiagramm dargestellt, wie viele Testfälle als Ergebnisse OK, Fehler oder einen Error ergaben. Die Ergebnisse OK und Fehler ergeben sich aus dem Soll-Ist-Vergleich. Bei einem Testfall kann es mehrere solcher Vergleiche und damit verbundener Ergebnisse geben. Im Bericht wird aber immer nur das schlimmste Ergebnis für den Testfall angezeigt. Die Einstufung erfolgt in der Reihenfolge OK, Fehler und Error. Der Error ergibt sich daraus, wenn ein Testfall nicht beendet werden konnte, zum Beispiel, wenn der Softwareroboter einen Button nicht erkennen oder erreichen und so den Prozess nicht fortsetzten kann [Anf. 16].

Über den Bericht kann auf die einzelnen Detailberichte der Testfälle navigiert werden. Darin werden alle Soll-Ist-Vergleiche eines Testfalles dargestellt sowie der Fortschritt und die Dauer angezeigt [Anf. 14].

Die erstellten XML- und HTML-Dateien werden gemeinsam je Testdurchlauf in einen Ablageordner abgelegt. Eine Archivierung der Ergebnisse ist daher durch das Archivieren des gesamten Ablageordners möglich [Anf. 17]. Dadurch, dass beim Archivieren auch alle XML-Dateien enthalten sind, ist in weiterer Folge ein Vergleich der Ergebnisse mit vergangenen Testdurchläufen möglich. Dazu muss vor dem Start des Testdurchlaufes im Framework der Ordner mit den archivierten Daten ausgewählt werden. Im Bericht werden dann zu den jeweiligen aktuellen Ergebnissen auch die historischen Ergebnisse angezeigt [Anf. 18].

Zusammenfassend sind somit alle im Abschnitt 2.1.6 angeführten Erfordernisse im Test-Reporting des Frameworks vorhanden. Die Anzahl der Regressionstests wie auch die Anzahl gegliedert nach Ergebnis, Ergebnisbeschreibung da die Soll- und Ist-Werte dargestellt werden, Informationen zur Lokalisierung der Fehler durch die Fortschrittsinformationen sowie eventuell auftretende Errors [Anf. 19].

3.4 Prozesse aus Sicht des Anwendenden

In diesem Abschnitt erfolgt die Beschreibung von zwei Prozessen aus Sicht des Anwendenden. Die Beschreibung erfolgt dabei anhand der erweiterten ereignisgesteuerten Prozesskette (eEPK). Dabei handelt es sich um den Prozess des Anlegens eines Testfalles und um die Auswahl der Testfälle sowie den Start des Testdurchlaufes.

3.4.1 Anlegen der Testfälle

Das Anlegen der Testfälle startet nach dem Erstellen der Softwareroboter für den jeweiligen Testfall. Ist dieser fertig, kann im Framework ein Formular geöffnet werden, mit dem ein neuer Testfall angelegt werden kann. In dieses Formular sind die aus Abschnitt 3.3.1 angeführten Informationen einzutragen. Zudem müssen auch die erforderlichen Testdaten in die Excel-Datei eingetragen und, sofern erforderlich, die weiteren für den Testfall erforderlichen Dateien in den Ordner kopiert werden. Nach dem vollständigen Erfassen der Daten und Speichern ist der Testfall angelegt. Abbildung 13 zeigt diesen Prozess anhand eines eEPK.

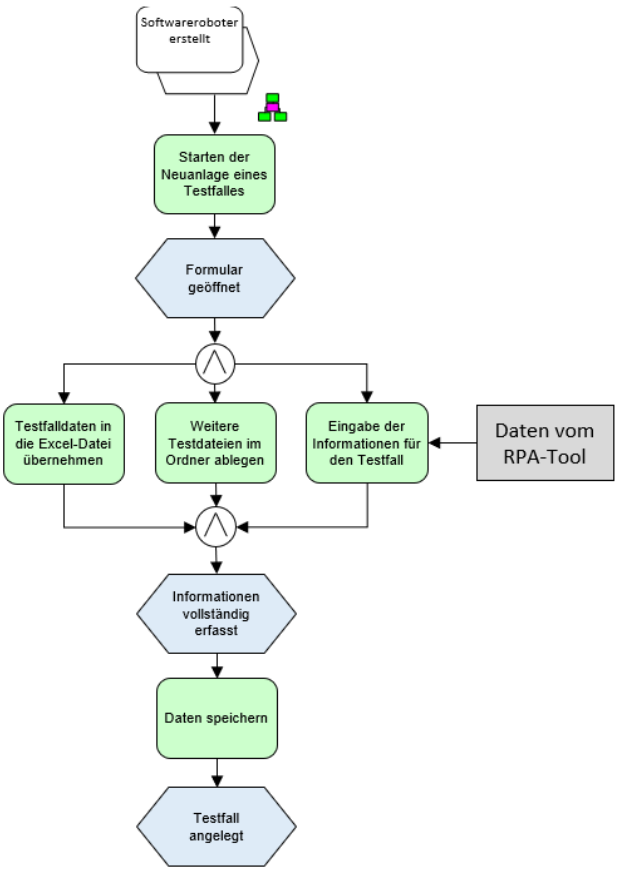


Abbildung 13: eEPK für das Anlegen der Testfälle im Framework (eigene Darstellung)

3.4.2 Auswahl der Testfälle und Start des Testdurchlaufes

Nachdem das Framework gestartet wurde, können die vom Anwendenden selbst definierten Bereiche und die darin enthaltenen Testfälle ausgewählt werden. Dabei können auch aus mehreren Bereichen die Testfälle gewählt werden [Anf. 11]. Die Reihenfolge der Testfälle ergibt sich aus der Reihenfolge wie der*die Anwender*in die Testfälle ausgewählt hat [Anf. 12]. Wurden alle Testfälle ausgewählt, wird vor dem Start des Testdurchlaufes noch eine Liste mit allen ausgewählten Testfällen in der Reihenfolge, wie diese beim Testdurchlauf abgearbeitet werden, geöffnet. In dieser Übersicht kann zum Vergleich ein altes Ergebnis ausgewählt werden und es ist möglich, eine Anmerkung für den Testdurchlauf einzutragen [Anf. 20]. Wurden alle Informationen eingetragen bzw. ausgewählt, kann der Testdurchlauf von dem*der Anwender*in gestartet werden. Abbildung 14 zeigt die Auswahl der Testfälle und den Start des Testdurchlaufes anhand eines eEPK.

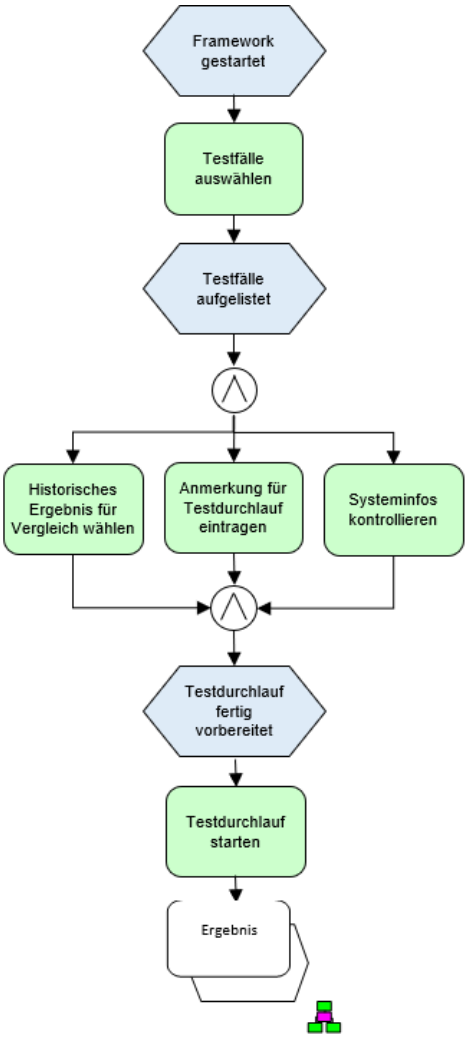


Abbildung 14: eEPK für die Auswahl der Testfälle und Start des Testdurchlaufes (eigene Darstellung)

3.5 Zusammenfassung

Zu Beginn dieses Kapitels wurden die Ziele und Nicht-Ziele für das Framework erarbeitet. Dabei wurden zwei wesentliche Punkte angeführt. Die Anwendung des Frameworks soll, wie auch bei der Entwicklung der Softwareroboter, ohne Programmierkenntnisse erfolgen und beim Framework handelt es sich nicht um eine voll funktionsfähige Software, sondern das Schwergewicht der Entwicklung liegt auf jenen Bereichen, die für die Beantwortung der Forschungsfrage wichtig sind. In einem nächsten Schritt wurden aus dem Kapitel Grundlagen und Stand der Technik die Anforderungen an das Framework abgeleitet. Die Anforderungen gliedern sich in drei Bereiche grundlegende Anforderungen, Anforderungen an die Steuerung der Softwareroboter und Anforderungen an das Test-Reporting. Auf Basis dieser Anforderungen erfolgte die Konzeption des Frameworks. Beginnend mit der Ausarbeitung der gesamten Architektur, wo auch die Einbindung der RPA-Software beschrieben wurde, gefolgt von der Beschreibung der Einbindung der Testfälle sowie der Steuerung der Softwareroboter und zuletzt wurde das Test-Reporting konzipiert. Am Ende wurden die beiden Prozesse Anlegen und die Auswahl eines Testfalles sowie der Start des Testdurchlaufes aus Sicht des Anwendenden in Form von einer eEPK beschrieben.

4 Prototypische Umsetzung dieses Frameworks

In diesem Kapitel erfolgt die Beschreibung der wesentlichen Komponenten des Frameworks bei der Umsetzung. Zu Beginn werden grundlegende Bereiche beschrieben, mit denen die Umsetzung des prototypischen Frameworks erfolgt. Danach erfolgt die Beschreibung der Schnittstellen, der Datenstruktur, der Benutzeroberfläche und Aufbau des Test-Reportings und Protokollierung.

4.1 Beschreibung Programmiersprachen

Die Umsetzung des Frameworks erfolgt in der Programmiersprache C#. Bei C# handelt es sich um eine von Microsoft entwickelte, moderne objektorientierte Programmiersprache. Mit dieser Programmiersprache können Anwendungen entwickelt werden, welche mit dem .NET Framework ausgeführt werden [Bill22].

Für die Entwicklung der Benutzeroberfläche kommt Windows Presentation Foundation (WPF) zum Einsatz. Dabei handelt es sich um ein von Microsoft zur Verfügung gestelltes Framework für Benutzeroberflächen, mit welchem Desktopanwendungen erstellt werden können. WPF bietet eine große Auswahl an Anwendungsentwicklungsfeatures und gehört zum .NET Framework [Terr00].

4.2 Entwicklungsumgebung für die Umsetzung des Prototyps

Die Entwicklung des Frameworks erfolgt primär in Visual Studio IDE. Dabei handelt es sich um ein Programm mit zahlreichen Funktionen, welche bei der Softwareentwicklung unterstützen. Visual Studio IDE ist eine von Microsoft zur Verfügung gestellte Entwicklungsumgebung, welche neben C# auch noch die Entwicklung mit weiteren Programmiersprachen unterstützt [Jmar00].

Für Entwicklungen mit den Skriptsprachen XSLT, HTML, Cascading Style Sheets (CSS) und JavaScript, welche für das Test-Reporting erforderlich sind, kommt Visual Studio Code zum Einsatz. Visual Studio Code ist ein Editor, welcher das Bearbeiten von Dateien durch Syntaxhervorhebung, Autovervollständigung und weiteren Funktionen unterstützt. Es ist eine Open Source Software, welche von Microsoft bereitgestellt wird und durch eine Vielzahl an Sprachkomponenten individuell erweitert werden kann [Docu00].

4.3 Schnittstellenbeschreibung

Dieser Abschnitt beschreibt die Erweiterungen, welche bei der RPA-Software erforderlich sind, damit die einzelnen Softwareroboter Informationen mit dem Framework austauschen und somit für die Testautomatisierung eingesetzt werden können. Die Erweiterungen umfassen das Holen der Daten aus der Testfall-XML, das Schreiben der Fortschritte und der Ergebnisse in die jeweilige Ergebnis-XML sowie das Schließen der RPA-Software. Abbildung 15 zeigt das Klassendiagramm für die Erweiterung der RPA-Software, damit sie im Zusammenhang mit dem Framework zur Testautomatisierung eingesetzt werden kann.

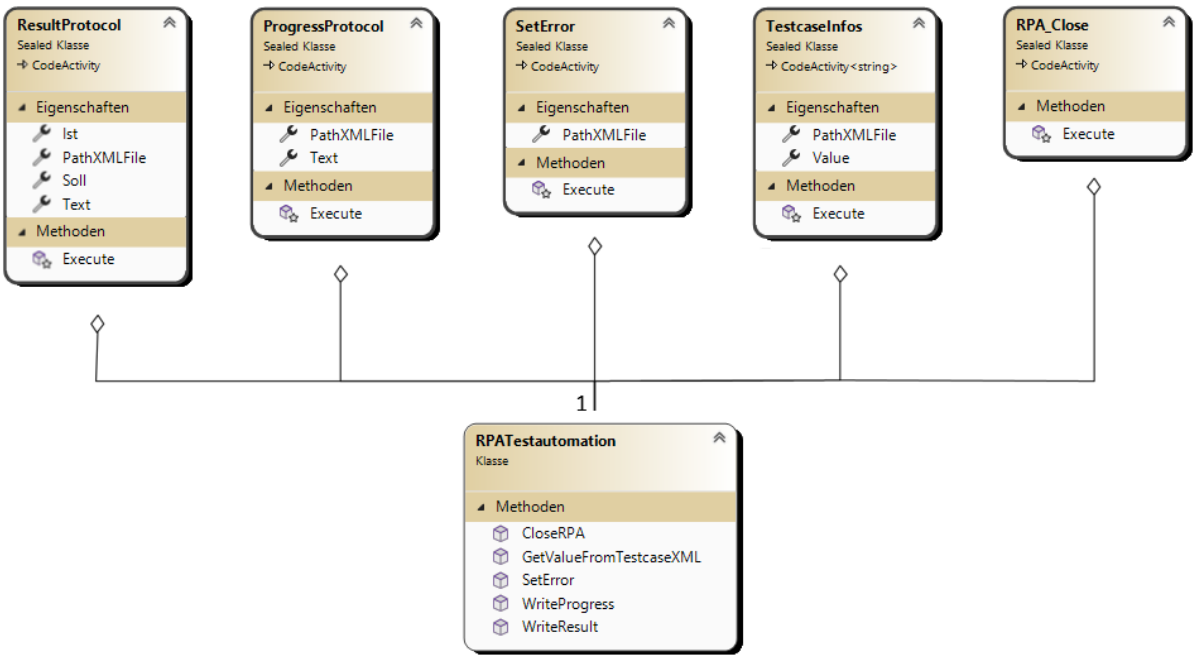


Abbildung 15: Klassendiagramm für die Erweiterung der RPA-Software. (eigene Darstellung)

All diese Klassen werden durch die Testautomation.dll zur Verfügung gestellt. Die Einbindung dieser Funktionen kann über zwei Arten erfolgen. Entweder als eine Windows-Workflow-Foundation-Aktivität oder direkt über den Aufruf der Funktionen aus der Klasse RPATestautomation. In der Abbildung 15 sind die oberen fünf Klassen die Aktivitäten und die untere Klasse beinhaltet die Funktionen. Jede Aktivität ruft über die Methode Execute die jeweilige zugehörige Methode aus der Klasse RPATestautomation auf und übergibt ihr die erforderlichen Parameter. Die erforderlichen Parameter sind bei den Methoden der Klasse RPATestautomation gleich wie bei der jeweiligen Aktivität.

RPA_Close – CloseRPA

Im Zuge der Entwicklung des Prototyps wurde festgestellt, dass sich nicht alle Softwareroboter am Ende des Prozesses automatisch oder durch eine eigene Funktion schließen lassen. Daher war die Zurverfügungstellung einer Funktion, welche die RPA-Software schließt, erforderlich.

TestcaseInfos – GetValueFromTestcaseXML

Die durch das Framework erstellten Informationen für den jeweiligen Testfall, in Form einer XML-Datei, sind zum Teil auch für den jeweiligen Softwareroboter von Interesse. Daher muss jede RPA-Software auf diese XML zugreifen und die jeweiligen Informationen auslesen können. Die Methode liefert immer nur einen Wert zurück, wodurch der Aufruf für jeden erforderlichen Wert notwendig ist.

SetError – SetError

Auch beim Softwareroboter kann es passieren, dass ein Fehler im Prozess auftritt und er den Testfall nicht bis zum Ende durchlaufen kann. In einem solchen Fall wird diese Methode aufgerufen und in der XML der Eintrag Error gesetzt. Diese Methode wird vor allem beim Abfangen von Ausnahmen durch Try and Catch eingesetzt.

ProgressProtocol – WriteProgress

Diese Methode fügt der XML im Ergebnisordner Fortschrittsinformationen hinzu. Diese können individuell durch die jeweiligen Anwendenden im Testprozess gesetzt und durch einen individuellen Text ergänzt werden. Auch diese Funktion kann beliebig oft durch den Softwareroboter aufgerufen werden.

ResultProtocol – WriteResult

Wird bei einem Testdurchlauf ein Testfall gestartet, wird durch das Framework im Ordner Ergebnisse eine XML mit dem Namen des Testfalles erstellt. Durch diese Methode wird dieser XML das Testergebnis Soll, Ist und ein Text hinzugefügt. Da es bei einem Testfall mehrere Testergebnisse geben kann, ist der Aufruf dieser Funktion beliebig oft möglich.

Der Sourcecode der Klasse RPATestautomation ist im Anhang C ersichtlich.

Die Zurverfügungstellung dieser Funktionen als Workflow-Foundation-Aktivität und als direkter Aufruf der Funktionen aus der Klasse RPATestautomation ist erforderlich, da nicht jede RPA-Software mit Aktivitäten oder dem direkten Aufruf einer Funktion aus einer Klasse umgehen kann. Die Einbindung von Aktivitäten ist bei UiPath und OpenRPA möglich. Dazu ist lediglich die Ablage der DLL in ein bestimmtes Verzeichnis erforderlich. Die Aktivitäten werden danach wie die anderen Funktionen in einer Liste angezeigt. Eine beispielhafte Darstellung der Erweiterungen in Form von Aktivitäten anhand von OpenRPA wird in Abbildung 16 gezeigt.

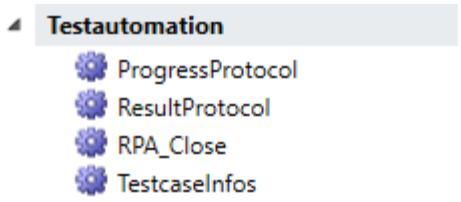


Abbildung 16: Erweiterungen in Form von Aktivitäten anhand von OpenRPA

Diese Funktionen können innerhalb der jeweiligen RPA-Software wie alle anderen bereits vorhandenen Funktionen ausgewählt und verwendet werden.

Bei taskt und Automation Anywhere ist die Einbindung durch eine Aktivität nicht möglich, aber sie können Methoden direkt aus Klassen aufrufen und verwenden. Abbildung 17 zeigt den Aufruf von Methoden aus einer Klasse in der DLL anhand der RPA-Software taskt.

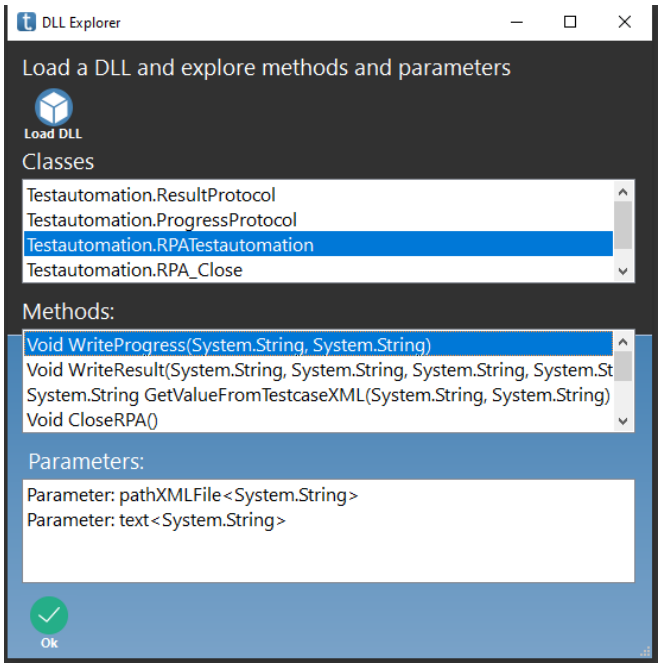


Abbildung 17: Aufruf von Methoden aus der DLL am Beispiel von taskt

In der Abbildung 17 sieht man die Auflistung aller in der DLL vorhandenen Klassen. Nach Auswahl der Klasse RPATestautomation werden die Methoden aufgelistet und je nach Auswahl die zugehörigen erforderlichen Parameter.

4.4 Benutzeroberfläche

In diesem Abschnitt wird für ein besseres Verständnis der Prozesse im Framework die Benutzeroberfläche des Frameworks vorgestellt. Die in der Benutzeroberfläche dargestellten Testfälle sind nur ein Muster und dienen der exemplarischen Darstellung.

4.4.1 Allgemein

In Abbildung 18 ist die Benutzeroberfläche dargestellt, welche sich in 3 Hauptbereiche gliedert. Die Menüleiste (1), der Navigationsbereich (2) und der Hauptbereich (3). In der Menüleiste kann der*die Anwender*in einen neuen Testfall anlegen, einen bestehenden bearbeiten, den Ordner mit den Testfällen bzw. den Ergebnissen öffnen oder den Testdurchlauf starten. Je nach Auswahl im Navigationsbereich sind die entsprechenden Schaltflächen aktiv.

Im Navigationsbereich kann zwischen verschiedenen Anzeigen für den Hauptbereich navigiert werden. Dabei stehen einem auch hier 3 Anzeigen zur Verfügung. Eine Übersicht über die im Framework vorhandenen Testfälle, die für einen Testdurchlauf ausgewählten Testfälle oder die Anzeige des entsprechenden Test-Reportings. Die Details zur Anzeige im Hauptbereich werden in den nächsten Abschnitten näher beschrieben.

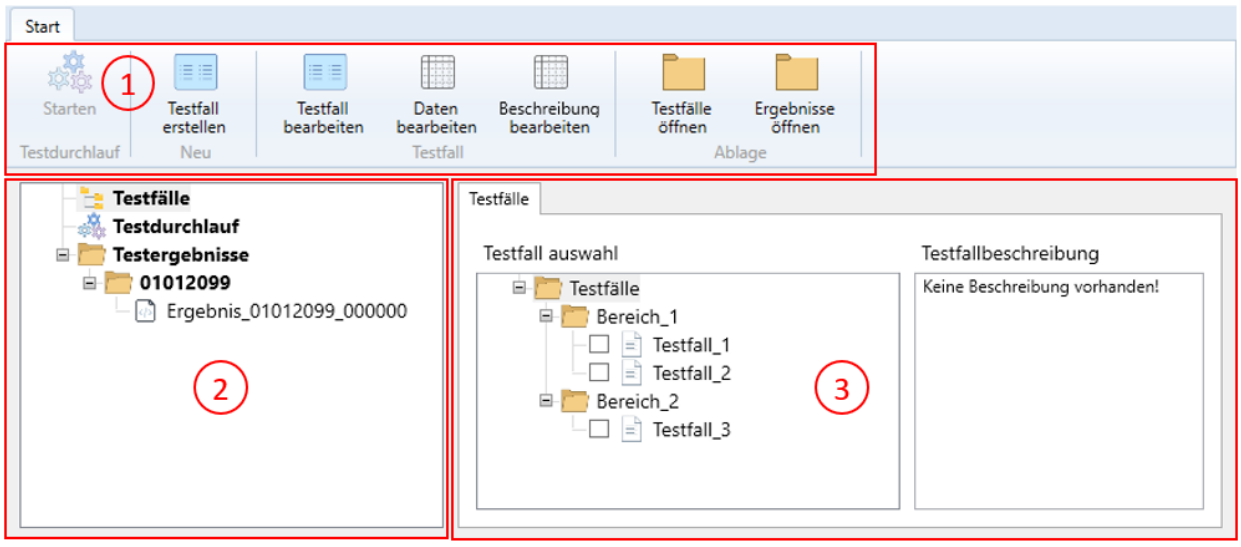


Abbildung 18: Allgemeine Darstellung der Benutzeroberfläche des Frameworks

4.4.2 Testfälle

Abbildung 19 zeigt die Anzeige der Testfälle im Hauptbereich. Die Anzeige der Testfälle umfasst den Strukturbaum mit den Bereichen und den darin enthaltenen Testfällen. Die Testfälle können über die Checkboxen von dem*der Anwender*in für einen Testdurchlauf ausgewählt werden. Dabei ist eine mehrfache Auswahl möglich. Auf der rechten Seite dieser Anzeige befindet sich die Testfallbeschreibung. Diese wird angezeigt, sobald ein Testfall angeklickt wurde. Bei diesem Feld für die Testfallbeschreibung handelt es sich um ein Richtextfeld, welches auch das Strukturieren von Texten zulässt.

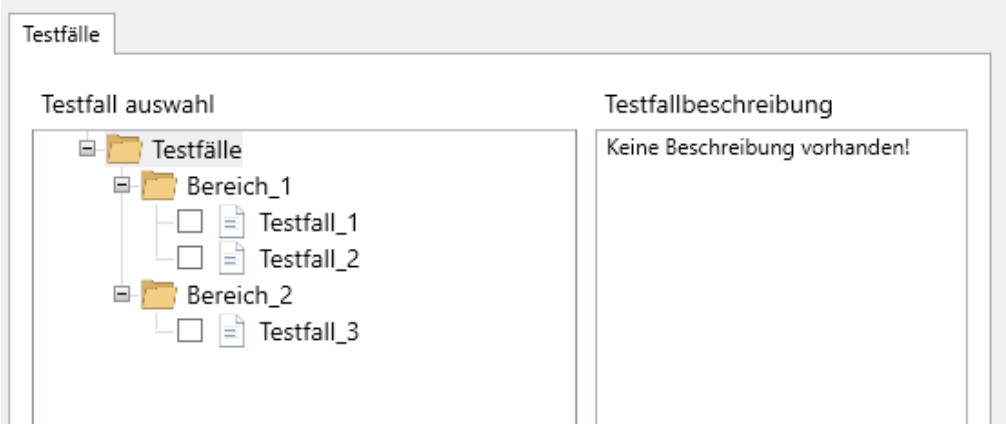


Abbildung 19: Anzeige der Testfälle in der Benutzeroberfläche

In Abbildung 20 sind die für das Anlegen und Bearbeiten der Testfälle erforderlichen Schaltflächen der Menüleiste dargestellt.

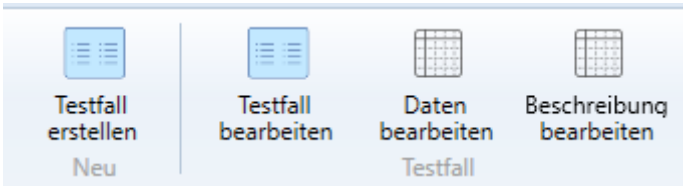


Abbildung 20: Schaltflächen für die Testfallbearbeitung

Bei den Schaltflächen „Testfall erstellen“ und „Testfall bearbeiten“ erscheint ein eigenes kleines Fenster für die Eingabe bzw. für die Bearbeitung der entsprechenden Daten für einen Testfall. Abbildung 21 zeigt das Eingabefenster mit den Steuerelementen.

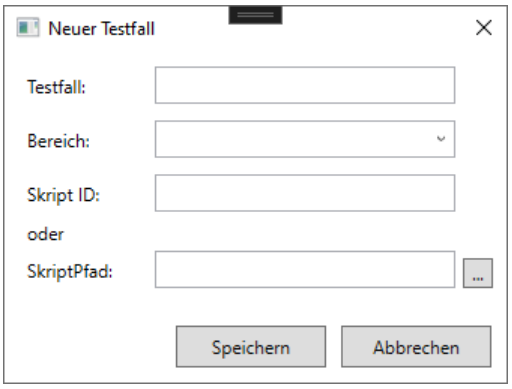


Abbildung 21: Eingabefenster für die Daten eines Testfalles

Bei der Anlage eines neuen Testfalles muss der Name eingegeben werden und ein bestehender Bereich ausgewählt oder durch Eingabe ein neuer Bereich angelegt werden. Zudem muss für den Testfall je nach verwendeter RPA-Software entweder die Skript ID oder der Skriptpfad angegeben werden. Durch die Schaltfläche „Speichern“ werden die Informationen zu diesem Testfall in der Testfallablage abgelegt.

Über die Schaltfläche „Daten bearbeiten“ können für den ausgewählten Testfall die entsprechenden Daten in Form einer Excel-Datei bearbeitet werden. Diese Daten werden im selben Verzeichnis wie der Testfall in Form einer Excel-Datei gespeichert. Die Testfallbeschreibung kann über die Schaltfläche „Beschreibung bearbeiten“ eingegeben oder bearbeitet werden. Auch hier ist es so, dass sich ein eigenes Fenster, wie in Abbildung 22 dargestellt, für die Bearbeitung der Testfallbeschreibung öffnet.

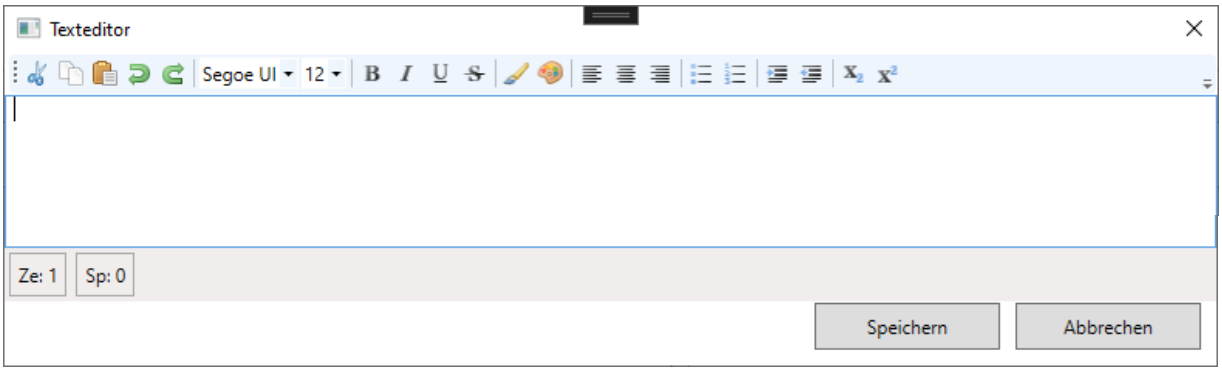


Abbildung 22: Fenster für die Eingabe oder die Bearbeitung der Testfallbeschreibung

Der Texteditor für die Testfallbeschreibung besteht aus einer Menüleiste, dem Eingabebereich, einer Informationsleiste und den Schaltflächen für das Speichern oder Abbrechen der Eingabe. Die Menüleiste beinhaltet eine kleine Auswahl an Möglichkeiten für das Bearbeiten vom Text, damit dieser in geeigneter Weise formatiert werden kann. Die Informationsleiste beinhaltet Angaben über die Zeile und Spalte, wo

sich aktuell die Position des Cursors im Text befindet. Das soll ein schnelleres Navigieren und Zurechtfinden im Text ermöglichen. Die Eingabe ist in Bezug auf die Anzahl der Zeichen nicht limitiert.

4.4.3 Testdurchlauf

Wird im Navigationsbereich Testdurchlauf ausgewählt, werden im Hauptbereich alle zuvor ausgewählten Testfälle aufgelistet. In Abbildung 23 ist eine beispielhafte Darstellung der Anzeige von ausgewählten Testfällen. Hier kann nochmal kontrolliert werden, ob für den Testdurchlauf alle Testfälle vorhanden sind. Im rechten Bereich der Anzeige kann für den Vergleich ein älterer Testdurchlauf ausgewählt werden. Es kann aber auch „kein Ergebnis“ ausgewählt werden und es erfolgt kein Vergleich der Ergebnisse mit einem älteren Testdurchlauf. Zudem befindet sich im rechten Bereich auch die Möglichkeit, dass der*die Anwender*in Anmerkungen zum Testdurchlauf eingeben kann und es werden dem*der Anwender*in Systeminformationen angezeigt. In Abbildung 23 werden die Systeminformationen rechts mit grauem Hintergrund angezeigt. Die Systeminformationen beinhalten Informationen zum*zur User*in, Betriebssystem, Arbeitsspeicher und Laufwerkinformationen. Die Anmerkungen und die Systeminformationen werden auch in die Übersicht des Test-Reportings übernommen.

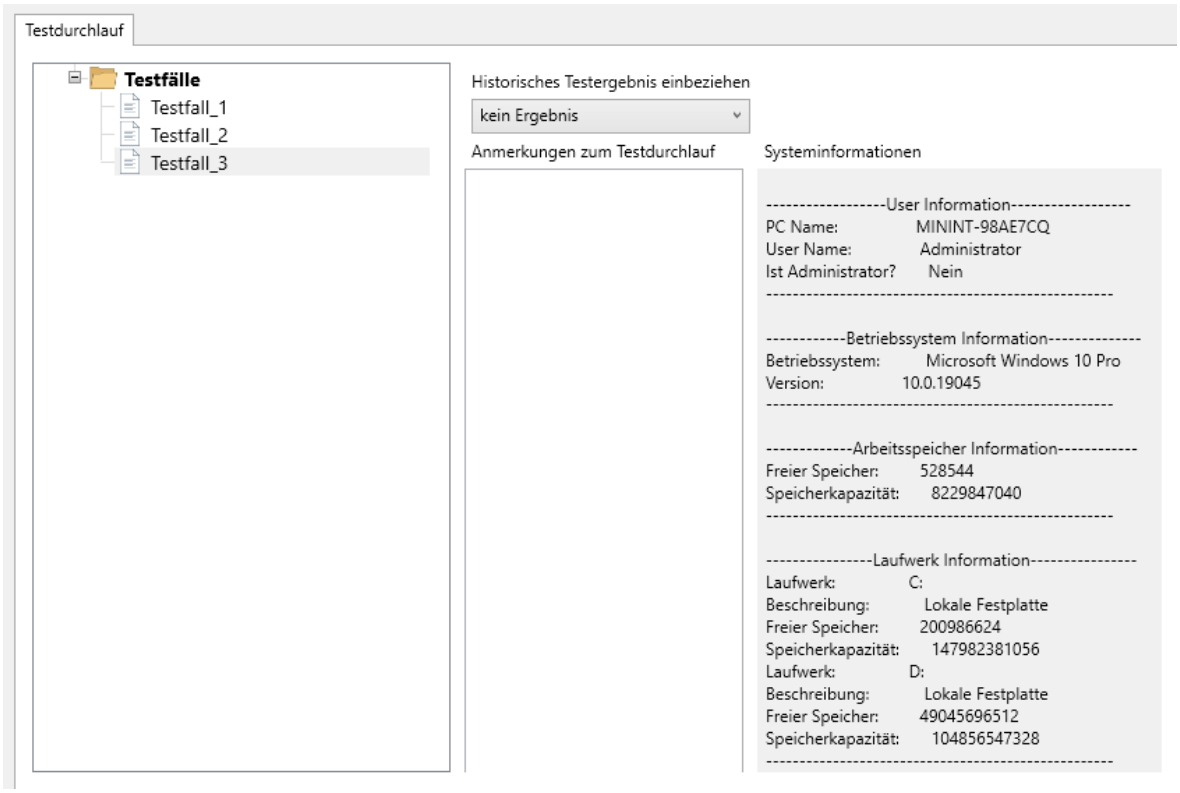


Abbildung 23: Informationen für den Testdurchlauf in der Benutzeroberfläche

Über die in Abbildung 24 dargestellte Schaltfläche, welche sich im Menüband befindet, wird der Testdurchlauf gestartet. Nach Betätigen dieser Schaltfläche wird das Framework automatisch minimiert, sodass es zu keinem Konflikt kommt, wenn der Softwareroboter seine Arbeit beginnt.

Abbildung 24: Schaltfläche für das Starten des Testdurchlaufes

Nachdem die Softwareroboter alle ausgewählten Testfälle abgearbeitet haben, wird das Framework wieder maximiert und die Ergebnisse vom Testdurchlauf werden angezeigt.

4.5 Beschreibung der Abläufe und Methoden für die Steuerung der Softwareroboter und des Test-Reportings

In diesem Abschnitt erfolgt die Beschreibung der wichtigsten Funktionen des Frameworks. Diese beinhalten die für den Betrieb des Frameworks erforderlichen Einstellungen sowie die Prozesse des Testdurchlaufes und die Erstellung des Test-Reportings.

4.5.1 Einstellungen

Damit das Framework eingesetzt werden kann, müssen zu Beginn einige grundlegende Einstellungen vorgenommen werden. Diese Einstellungen erfolgen bei den Prototypen des Frameworks direkt in der Registry. Die Einstellungen werden unter folgendem Pfad in der Registry gespeichert:

HKEY_CURRENT_USER\SOFTWARE\Framework_RPA

Beim erstmaligen Start des Frameworks werden die Einträge mit den Grundwerten erstellt. In Abbildung 25 ist eine Übersicht aller möglichen Einstellungen in der Registry für das Framework angeführt.

Abbildung 25: Übersicht der erforderlichen Einstellungen in der Registry

Argument_SkriptID und Argument_SkriptPfad

Bei den Einträgen Argument_SkriptID und Argument_SkriptPfad handelt es sich um die Übergabeparameterbezeichnungen, welche für den Start des jeweiligen Softwareroboters erforderlich sind. So müsste beispielsweise für den Start eines Softwareroboters mit OpenRPA beim Argument_SkriptID der Eintrag –workflowid gesetzt werden. Auf der Seite 36 in Abbildung 11 ist ein solches Argument für den Start von OpenRPA angeführt. Da nicht jede RPA-Software mit SkriptID oder SkriptPfad gestartet werden kann, ist nur bei dem jeweiligen notwendigen Eintrag ein entsprechender Wert zu setzen.

Max_Dauer

Hier ist die Dauer in Millisekunden einzutragen, wie lange das Framework wartet, bis der Softwareroboter seinen Prozess abgeschlossen hat. Der Grundwert liegt hier bei 900000. Das entspricht einer Wartezeit von 15 Minuten. Das bedeutet, wenn der Softwareroboter bei einem Testfall ein Problem hat und den Testfall nicht erfolgreich beenden kann, wird der Softwareroboter nach dieser Zeit eliminiert. Das Framework setzt danach automatisch beim Testfallergebnis den Wert Error.

Pfad_Ergebnisse

Bei diesem Eintrag wird der Pfad angegeben, wo die Ergebnisse abgelegt werden sollen. Die Ergebnisse beinhalten das gesamte Test-Reporting. Dadurch kann der*die Anwender*in das Framework individuell an die Verzeichnisstruktur anpassen.

Pfad_Testfälle

In diesem Eintrag wird der Pfad für die Ablage der Testfälle angegeben. Alle neu angelegten Testfälle werden in diesem Pfad abgelegt bzw. die im Framework angezeigten Testfälle werden von diesem Pfad geladen. Hier können die Testfälle auch

auf einem Netzlaufwerk abgelegt werden, das ermöglicht die zentrale Verwaltung der Testfälle innerhalb eines Entwicklerteams.

RPA_Software

Da das Framework mit unterschiedlichen RPA-Tools verwendet werden kann, müssen in diesem Eintrag der Pfad und die Bezeichnung der ausführbaren Datei der RPA-Software angegeben werden.

4.5.2 Testdurchlauf und Steuerung der Softwareroboter

Für den Testdurchlauf werden alle ausgewählten Testfälle in einer Schleife durchlaufen und abgearbeitet. In Abbildung 26 ist das Sequenzdiagramm für das Ausführen der einzelnen Testfälle dargestellt.

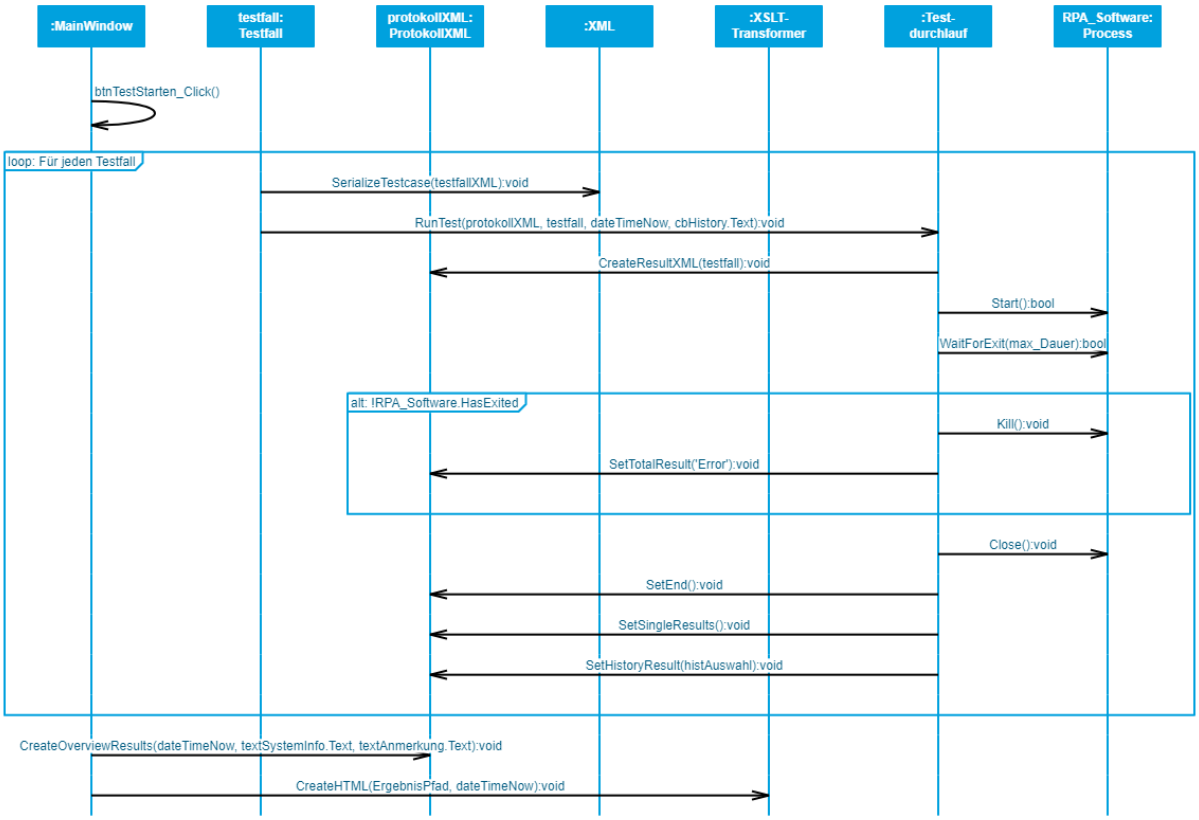


Abbildung 26: UML-Sequenzdiagramm für das Ausführen der einzelnen Testfälle

Die von dem*der Anwender*in ausgewählten Testfälle werden durch die Betätigung der Schaltfläche „Testdurchlauf Starten“ und in weiterer Folge der Methode btnTeststarten_Click() ausgeführt. Danach wird eine Schleife für jeden ausgewählten Testfall durchlaufen. In dieser Schleife werden zu Beginn die Daten zum Testfall, welche in der XML gespeichert sind, durch den Ergebnisnamen ergänzt. Die Ergebnisse werden

immer im Ergebnispfad in einem Ordner mit dem aktuellen Datum gelegt, daher ist es erforderlich, diese Daten in der XML des Testfalles zu ergänzen. Zudem werden diese Daten auch vom Softwareroboter benötigt und aus dieser XML ausgelesen. Das erfolgt durch die Methode SerializeTestcase() der statischen Klasse XML. Danach erfolgt der Aufruf der Methode RunTest() der statischen Klasse Testdurchlauf, welche den Startprozess auslöst. Bevor der Softwareroboter gestartet werden kann, ist noch die Erstellung der XML für das Ergebnis mit den jeweiligen Basisdaten für den Testfall erforderlich. Das erfolgt durch die Methode CreateResultXML() der Klasse ProtokollXML. Danach wird das Objekt RPA_Software aus der Klasse Process erstellt. Diesem Objekt werden die für den Start des Softwareroboters erforderlichen Startargumente zugeordnet und mit der Methode Start() wird der Softwareroboter gestartet. Durch den Aufruf der Methode WaitForExit(max_Dauer) wartet das Framework, bis der Softwareroboter seinen Prozess beendet hat. Zudem werden dieser Methode die in den Einstellungen festgelegten Millisekunden übergeben, wodurch das Warten nach Ablauf dieser Zeit automatisch beendet wird. Im Anschluss wird durch den Aufruf der Eigenschaft HasExited des Objektes RPA_Software geprüft, ob der Prozess beendet wurde oder ob die Zeit abgelaufen ist. Ist die Zeit abgelaufen und das Framework hat das Warten beendet, müssen der Prozess und damit verbunden der Softwareroboter auch beendet werden. Das erfolgt durch den Aufruf von Kill(). Zudem wird durch die Methode SetTotalResult() in die XML für das Ergebnis der Wert „Error“ eingetragen, da davon ausgegangen werden kann, dass der Prozess zu einem Fehler geführt hat, wenn sich der Softwareroboter nach einer entsprechenden Zeit nicht selbst beendet hat. Egal wie der Prozess beendet wurde, müssen die dem Prozess zugeordneten Ressourcen wieder freigegeben werden, was durch den Aufruf der Methode Close() erfolgt. Somit ist die Steuerung der Softwareroboter beendet und es müssen durch das Framework nur noch abschließende Eintragungen in die XML mit den Ergebnissen erfolgen. Als erstes wird der Zeitpunkt für das Ende durch die Methode SetEnd() eingetragen. Die Methode SetSingleResult() wertet die einzelnen Ergebnisse aus und ergänzt diese durch das Attribute Result, wo OK oder Fail eingetragen wird. Die letzte Methode, welche in der Schleife aufgerufen wird, ist SetHistoryResult(), wodurch das Gesamtergebnis eines ausgewählten vergangenen Testdurchlaufes eingetragen wird. Wurden so alle Testfälle durchlaufen, erfolgt am Ende die Erstellung des Gesamtergebnisses durch die Methode CreateOverviewResults(). Dabei wird aus den einzelnen XML mit den Ergebnissen der jeweiligen Testfälle eine XML als Gesamtergebnis mit den Informationen für die Übersicht beim Test-Reporting erstellt. Am Ende wird die Methode CreateHTML() der

statischen Klasse XSLTTransformer aufgerufen, welche alle XML-Dateien in HTML-Dateien transformiert und in die Ergebnisablage speichert.

In Abbildung 27 ist das Klassendiagramm mit den Klassen, welche für den Testdurchlauf erstellt wurden und erforderlich sind.

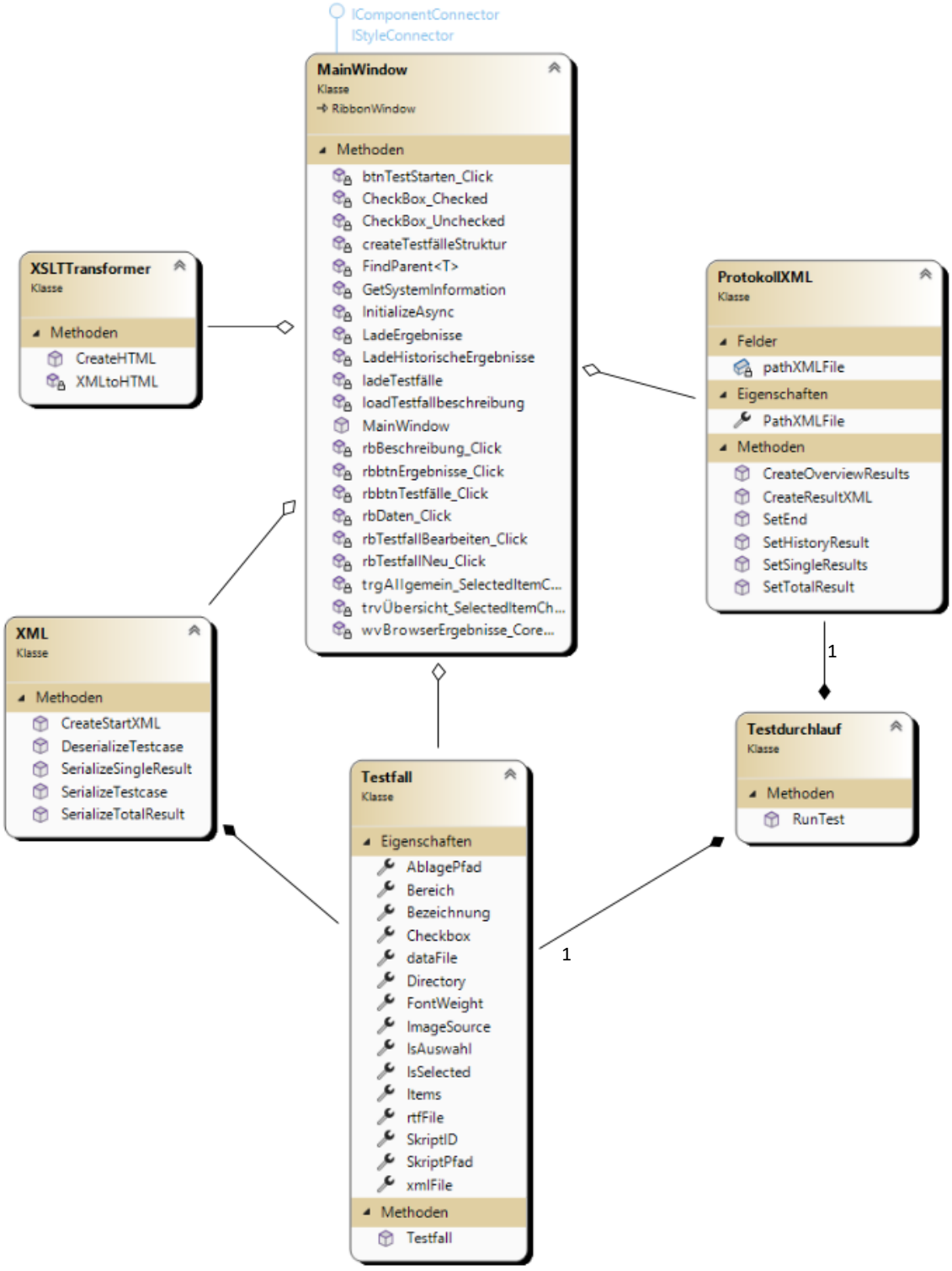


Abbildung 27: Klassendiagramm mit den Klassen für einen Testdurchlauf.

4.5.3 Test-Reporting erstellen

Die Erstellung des Test-Reportings erfolgt durch die Transformation der XML-Dateien durch XSLT-Stylesheets zu HTML-Dateien. Durch den Aufruf der Methode CreateHTML() startet die Erstellung der Ergebnisdokumente. Dabei werden alle XML-Dateien im Ergebnisordner durchlaufen. Jede Datei wird geprüft, ob sie die XML für das Gesamtergebnis ist oder nicht. Diese Unterscheidung ist wichtig, da der Methode XMLtoHTML() das jeweilige richtige XSLT-Stylesheet übergeben werden muss. Nach erfolgreicher Transformation bleiben die XML gemeinsam mit der HTML in der Ergebnisablage. Die XML werden nicht gelöscht, da sie bei zukünftigen Testdurchläufen für die Einbindung der historischen Ergebnisse weiterhin benötigt werden. Das in Abbildung 28 dargestellte Sequenzdiagramm verdeutlicht die Abläufe für die Erstellung des Test-Reportings.

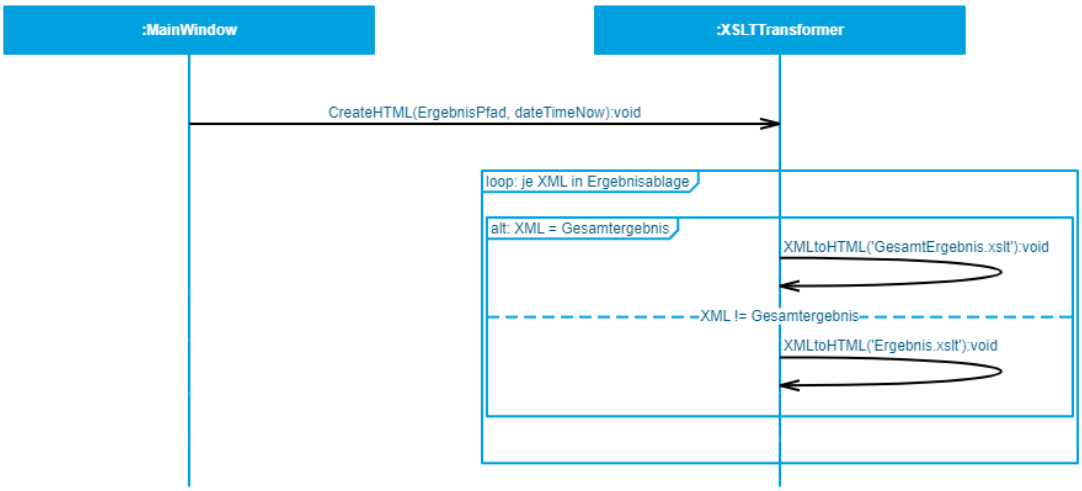


Abbildung 28: UML-Sequenzdiagramm für das Erstellen des Test-Reportings

Im Anhang D ist das XSLT-Stylesheet Ergebnis.xslt angeführt, welches die Detailergebnisse im Test-Reporting erstellt.

Im Anhang E ist das XSLT-Stylesheet GesamtErgebnis.xslt angeführt, mit dem die Übersicht im Test-Reporting erstellt wird.

4.6 Aufbau Test-Reporting und Protokollierung

Dieser Abschnitt beschreibt den Aufbau und die Struktur des Test-Reportings. Zuerst wird die Übersicht beschrieben und danach die Detailergebnisse, welche für jeden Testfall erstellt werden.

4.6.1 Test-Reporting Übersicht

Im Navigationsbereich (Seite 47, Abbildung 18, Bereich 2) werden alle im Ergebnisordner abgelegten Testergebnisse aufgelistet. Die Auflistung erfolgt gegliedert nach Datum. Wenn der*die Anwender*in aus dem Navigationsbereich heraus ein Test-Reporting im Framework öffnet, gelangt er oder sie direkt zur Übersicht. Abbildung 29 zeigt eine beispielhafte Übersicht eines Testdurchlaufes.

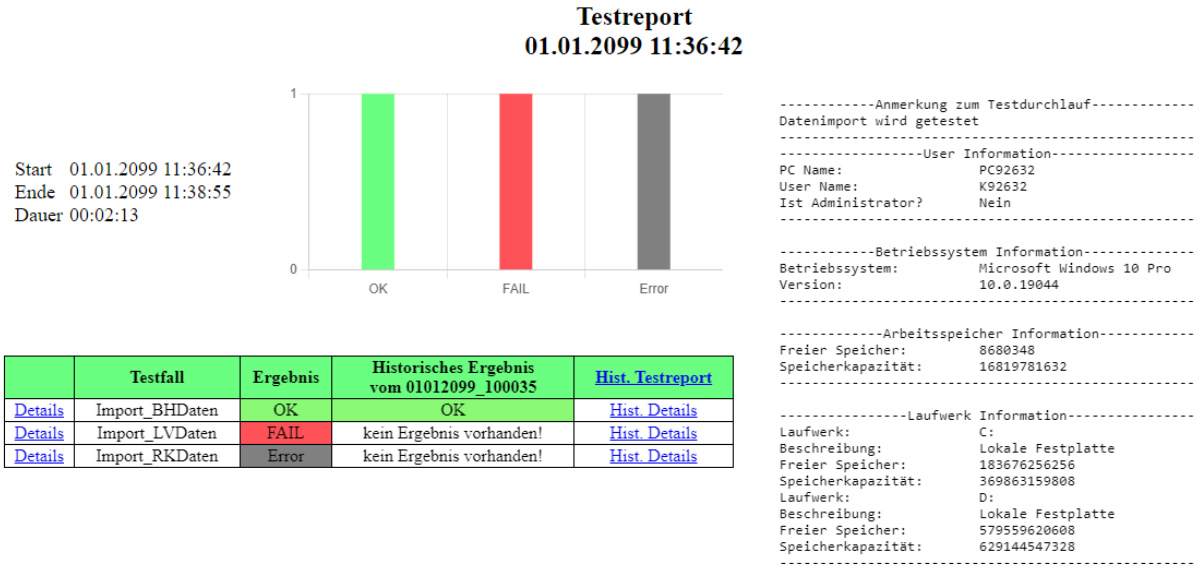


Abbildung 29: Ergebnisse nach einem Testdurchlauf im Test-Reporting

Auf der linken Seite ist der Startzeitpunkt und das Ende mit jeweils dem Datum und der Uhrzeit angeführt. Darunter steht die berechnete Dauer im Format Stunden:Minuten:Sekunden.

Rechts daneben ist die Übersicht über die einzelnen Ergebnisse als Balkendiagramm dargestellt. Auf der y-Achse ist die Anzahl und auf der x-Achse sind die Ergebnisse zu sehen. Die Balken werden zudem auch noch in den Farben Grün für OK, Rot für FAIL und Grau für Error angezeigt.

Darunter sind die einzelnen Testfälle mit ihrer Bezeichnung und dem jeweiligen Ergebnis angeführt. In der ersten Spalte sind bei jedem Testfall Hyperlinks mit der Bezeichnung Details. Über diese Links kann direkt zu den Detailergebnissen des jeweiligen Testfalles navigiert werden. Wurde vor dem Testdurchlauf ausgewählt, dass dieses Ergebnis mit einem Ergebnis aus der Vergangenheit verglichen werden soll, dann sind neben der Spalte Ergebnis zwei weitere Spalten für das historische Ergebnis zu sehen. Direkt neben den Ergebnissen wird die Spalte Historisches Ergebnis vom

TTMMJJJJ_hhmmss angezeigt. Als letzte Spalte sind wieder Hyperlinks, aber diesmal zu den jeweiligen historischen Details. Zudem kann über den Hyperlink in der Spaltenüberschrift zum historischen Test-Reporting navigiert werden.

Auf der rechten Seite werden die von dem*der Anwender*in eingetragenen Anmerkungen zum Testfall angezeigt und mehrere ausgewählte Systeminformationen werden aufgelistet.

4.6.2 Test-Reporting Protokollierung

Abbildung 30 zeigt eine beispielhafte Darstellung eines Detailergebnisses zu einem Testfall.

zurück

Datenimport
Import_BHDaten

Start 01.01.2099 11:47:11
Ende 01.01.2099 11:48:06
Dauer 00:00:55

Ergebnisse

TimeStamp	Hinweis	Soll	Ist	Ergebnis
01.01.2099 11:48:05	Datei vorhanden	def_AVZ_2016.FIL	def_AVZ_2016.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_AVZ_2017.FIL	def_AVZ_2017.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_AVZ_2018.FIL	def_AVZ_2018.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_GKontenliste.FIL	def_GKontenliste.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Journal_2016.FIL	def_Journal_2016.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Journal_2017.FIL	def_Journal_2017.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Journal_2018.FIL	def_Journal_2018.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Konten_2016.FIL	def_Konten_2016.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Konten_2017.FIL	def_Konten_2017.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Konten_2018.FIL	def_Konten_2018.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Kontenliste.FIL	def_Kontenliste.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Saldenliste_2016.FIL	def_Saldenliste_2016.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Saldenliste_2017.FIL	def_Saldenliste_2017.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	def_Saldenliste_2018.FIL	def_Saldenliste_2018.FIL	OK
01.01.2099 11:48:05	Datei vorhanden	EinleseLOG.LIX	EinleseLOG.LIX	OK
01.01.2099 11:48:05	Datei vorhanden	EinleseLOG.LOG	EinleseLOG.LOG	OK
01.01.2099 11:48:05	Datei vorhanden	Einlesen.aclscript	Einlesen.aclscript	OK
01.01.2099 11:48:05	Datei vorhanden	IndexGKontenListe.INX	IndexGKontenListe.INX	OK
01.01.2099 11:48:05	Datei vorhanden	IndexKontenListe.INX	IndexKontenListe.INX	OK
01.01.2099 11:48:05	Datei vorhanden	RZLAVZ01.WSP	RZLAVZ01.WSP	OK
01.01.2099 11:48:05	Datei vorhanden	RZLJournal07.WSP	RZLJournal07.WSP	OK
01.01.2099 11:48:05	Datei vorhanden	RZLKonten04.WSP	RZLKonten04.WSP	OK
01.01.2099 11:48:05	Datei vorhanden	RZLSL03.WSP	RZLSL03.WSP	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Journal_2016.FIL	SL_def_Journal_2016.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Journal_2017.FIL	SL_def_Journal_2017.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Journal_2018.FIL	SL_def_Journal_2018.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Konten_2016.FIL	SL_def_Konten_2016.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Konten_2017.FIL	SL_def_Konten_2017.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	SL_def_Konten_2018.FIL	SL_def_Konten_2018.FIL	OK
01.01.2099 11:48:06	Datei vorhanden	Start.aclscript	Start.aclscript	OK

Protokoll

TimeStamp	Hinweis
01.01.2099 11:47:22	Testfallinfos geladen!
01.01.2099 11:47:24	Testfalldaten aus Excel geladen
01.01.2099 11:47:24	Datein für den Import in das ACL Verzeichnis kopiert
01.01.2099 11:47:24	ACL gestartet
01.01.2099 11:47:37	Neues Projekt erstellt
01.01.2099 11:47:40	Datenauswahl Fenster
01.01.2099 11:47:43	Produktauswahl Fenster
01.01.2099 11:47:44	Tabellenübersicht Fenster
01.01.2099 11:47:48	Letztes Fenster Fenster
01.01.2099 11:48:05	Import beendet

Abbildung 30: Detailergebnisse zu einem Testfall

Ganz oben ist die Überschrift zu sehen. Diese besteht aus dem Bereich, zu jenem der Testfall gehört, und der Testfallbezeichnung. Auch hier wird wieder der Startzeitpunkt und das Ende sowie die Dauer angezeigt. Diese bezieht sich hier aber nur auf den jeweiligen Testfall. Darunter werden zwei Tabellen dargestellt.

In der linken Tabelle werden die Ergebnisse des Soll-Ist-Vergleiches angeführt. Diese bestehen aus dem Zeitpunkt, wann der Vergleich durchgeführt wurde, einem Hinweis zum Vergleich, den Soll- und den Ist-Werten sowie dem Ergebnis dieses Soll-Ist-Vergleiches. In der rechten Tabelle wird ein Fortschrittsprotokoll angezeigt. Dieses

besteht wieder aus dem Zeitpunkt, wann das Ereignis eingetreten ist und einem Hinweis, was zu diesem Zeitpunkt geschehen ist.

Über die Schaltfläche „Zurück“, links oben, kann wieder zur Übersicht zurück navigiert werden.

4.7 Zusammenfassung

Damit RPA-Software für die Testautomatisierung verwendet werden kann, muss diese um jene Funktionen erweitert werden, welche für die Testautomatisierung erforderlich sind. Der Softwareroboter muss in der Lage sein, die Ergebnisse sowie den Fortschritt zu dokumentieren und die jeweiligen Testfall-Informationen entsprechend abzufragen. Diese Erweiterungen werden durch das Framework als DLL zur Verfügung gestellt. Das Framework selbst ist eine Desktopanwendung und besitzt eine Benutzeroberfläche. Diese unterstützt den*die Anwender*in beim Anlegen, Bearbeiten der Testfälle und beim Durchführen der Testdurchläufe. Im Framework werden alle erstellten Testfälle aufgelistet und können mittels Checkbox ausgewählt werden. Zu jedem Testfall kann auch eine Beschreibung hinzugefügt werden, welche dem*der Anwendenden nach Klick auf den Testfall angezeigt wird. Zudem werden am Ende des Testdurchlaufes die Ergebnisse angezeigt. Diese werden in Form einer Übersicht und als Detailergebnis dargestellt.

5 Evaluierung des Prototyps

In diesem Kapitel erfolgt die Implementierung von Test-Cases in den Prototyp und die praktische Anwendung des Frameworks zur Evaluierung. Die Evaluierung des Prototyps erfolgt auf zwei unterschiedliche Weisen. Einerseits wird die Funktionsweise durch die Anwendung von Test-Cases auf ihre Richtigkeit geprüft und andererseits wird durch die praktische Anwendung des Frameworks geprüft, ob die Testautomatisierung anhand des Testprozesses von Seite 10 Abbildung 2 mit dem Framework umgesetzt werden kann. Dabei wird zuerst die Vorgehensweise bei der Auswahl geeigneter Testfälle sowie bei der Entwicklung der Softwareroboter beschrieben. Danach erfolgt die Beschreibung des Testobjektes und der ausgewählten Testfälle. Im Anschluss werden die Tätigkeiten der Softwareroboter beschrieben und gezeigt, wie die Entwicklung des Softwareroboters in der jeweiligen RPA-Entwicklungsumgebung aussieht. Danach wird der zweite Teil der Evaluierung, die praktische Anwendung des Frameworks, beschrieben. Am Ende erfolgt eine Zusammenfassung der Ergebnisse der Evaluierung, die Vor- und Nachteile des Frameworks beim Einsatz zur Testautomatisierung werden aufgezeigt.

5.1 Vorgehensweise bei der Implementierung

Der Prototyp des Frameworks und die RPA-Software OpenRPA sowie taskt wurden auf einem Notebook, welches für Softwaretests verwendet wird, installiert und eingerichtet. Das Notebook hat dieselbe Konfiguration wie die Produktivumgebung, in der die Anwender*innen die Software verwenden. Auf diesem Notebook werden bereits jetzt schon manuelle Tests durchgeführt.

Im nächsten Schritt wurden die bereits bestehenden, aber manuell durchgeführten Testfälle beurteilt, ob diese durch die Testautomatisierung ersetzt werden können. Dabei wurden Kriterien wie Häufigkeit der Ausführung und Wichtigkeit des Testprozesses, die Machbarkeit der Testautomatisierung mit RPA für die Beurteilung herangezogen. Aus den daraus resultierenden Testfällen wurden drei Testfälle für die Umsetzung ausgewählt. Danach erfolgte die Entwicklung des Softwareroboters. Für die Entwicklung dieses Softwareroboters wurde eine inkrementelle Vorgehensweise gewählt. Dabei wurde der Softwareroboter entwickelt und danach immer wieder selbst getestet, angepasst und stabilisiert. Dadurch soll vermieden werden, dass es zu einem Fehleralarm kommt und damit verbunden eine langwierige Fehlersuche beim

Testobjekt. In Abbildung 31 ist diese Vorgehensweise bei der Entwicklung des Softwareroboters schematisch dargestellt.

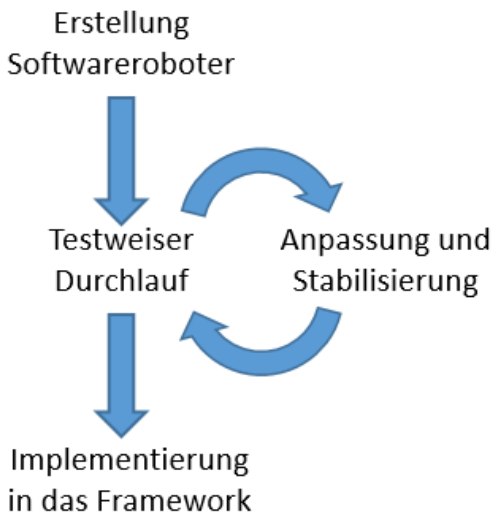


Abbildung 31: Vorgehensweise bei der Entwicklung des Softwareroboters (Eigene Darstellung in Anlehnung an [BBSG15])

Erst nach mehrmaligem Durchlauf wurde der jeweilige Softwareroboter, wie im Abschnitt 3.4.1 beschrieben, in das Framework integriert. Die integrierten Softwareroboter können mit unterschiedlichen Testdaten verwendet werden, wodurch sich mehrere eigene Testfälle ergeben. Mit dieser Vorgehensweise wurde ein Softwareroboter mit OpenRPA und einer mit taskt entwickelt und integriert. In Abbildung 32 sind die fertigen integrierten Testfälle im Framework dargestellt. Oben sind die Testfälle, welche mit OpenRPA ausgeführt werden können, und unten jene, die mit der RPA-Software taskt verwendet werden können.

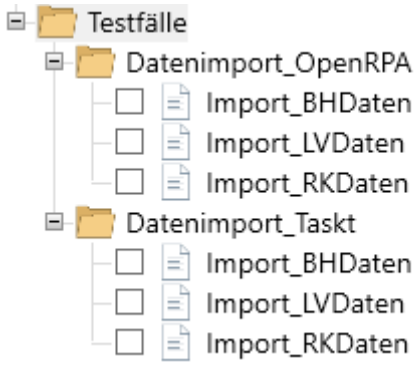


Abbildung 32: Fertig implementierte Softwareroboter im Framework

5.2 Beschreibung des Testobjektes

Das Testobjekt ist ein selbstentwickelter Datenimportassistent für die Software ACL Analytics. ACL Analytics ist eine Software, welche für Datenanalysen eingesetzt wird [Wasi00]. Mit diesem Datenimportassistenten können Datenexporte mit unterschiedlichen Formaten und aus unterschiedlichsten Softwareprogrammen importiert und einheitlich aufbereitet werden. Bei diesen Datenexporten handelt es sich zum Großteil um Buchhaltungsdaten, Kassendaten, Zeitaufzeichnungen oder sonstigen betrieblichen Grundaufzeichnungen in digitaler Form, welche für eine Datenanalyse im Rahmen eines Audits relevant sein können. Da diese Daten kein einheitliches Format besitzen, wurde dieser Datenimportassistent entwickelt und muss auch ständig an die digitalen Neuerungen angepasst werden.

Werden Daten mit diesem Datenimportassistenten importiert, müssen folgende Schritte durchgeführt werden.

- ACL-Projekt auswählen oder ein neues erstellen
- Daten auswählen
- Produkt auswählen
- Dateien prüfen
- Tabellenname anpassen
- Einlesen starten

Jeder dieser Schritte erfolgt über eine eigene Ansicht in der Benutzeroberfläche, welche zur Unterstützung der*des Anwendenden dient.

ACL-Projekt auswählen oder ein neues erstellen
Zu Beginn muss der*die Anwender*in ein ACL-Projekt auswählen, in welches die Daten importiert werden soll. In Abbildung 33 ist das Einstiegsfenster für den Datenimportassistenten dargestellt. Hier kann entweder ein bestehendes oder ein neues ACL-Projekt ausgewählt werden. Die Auswahl erfolgt durch die Angabe des Pfades zum ACL-Projekt. Dabei wird der*die Anwender*in durch einen Filedialog unterstützt.

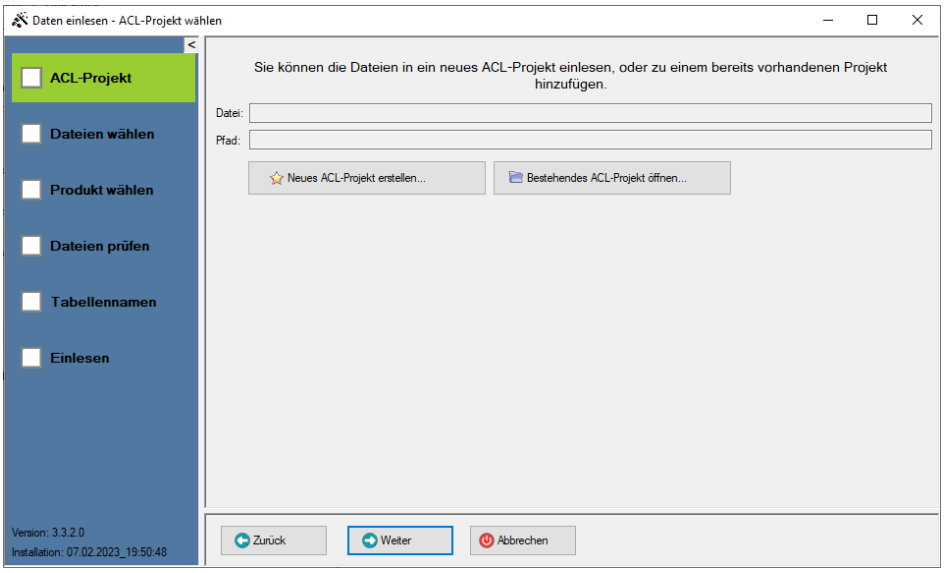


Abbildung 33: Datenimportassistent: Ansicht – ACL-Projekt

Daten auswählen

Im nächsten Schritt müssen die zu importierenden Dateien ausgewählt werden. Abbildung 34 zeigt die Ansicht für die Datenauswahl. Die Auswahl erfolgt durch Navigation in einem Treeview im linken Bereich der Ansicht zum Speicherplatz der Dateien. Danach werden im rechten Bereich die Dateien aufgelistet, welche sich im ausgewählten Ordner befinden. Mittels Drag and Drop werden Dateien in den unteren Bereich verschoben und somit für den Datenimport ausgewählt.

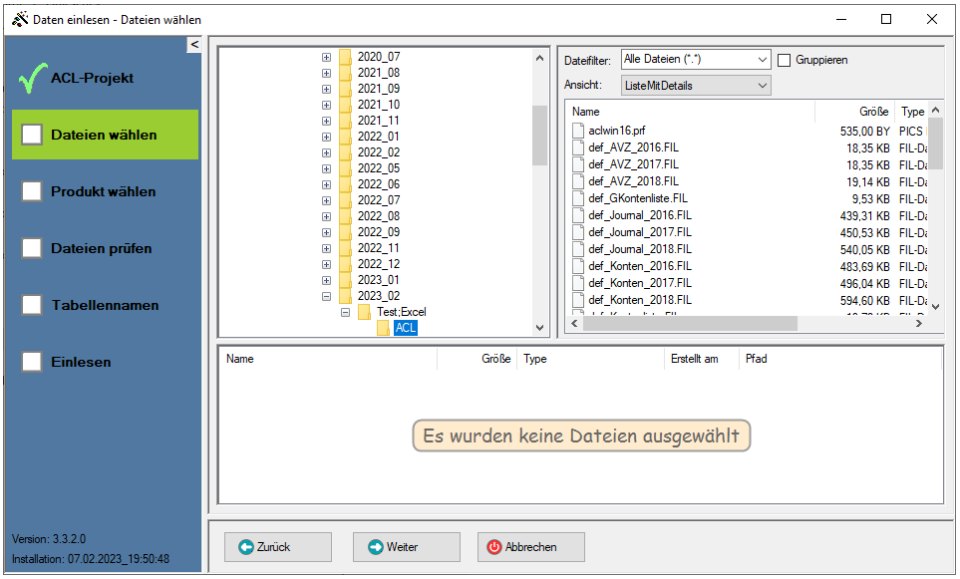


Abbildung 34: Datenimportassistent: Ansicht – Daten auswählen

Produkt auswählen

Nach Auswahl der Daten muss ein Produkt ausgewählt werden. Bei dem Produkt handelt es sich um die Bezeichnung des Softwareprogrammes, aus welchem die Daten exportiert wurden, zum Beispiel SAP. Abbildung 35 zeigt die Ansicht für die Produktauswahl. Oben kann nach verschiedenen Bereichen oder nach einzelnen Produkten gefiltert werden. Über die Checkboxen bei der Auflistung der Produkte in der Mitte wird das entsprechende Produkt ausgewählt.

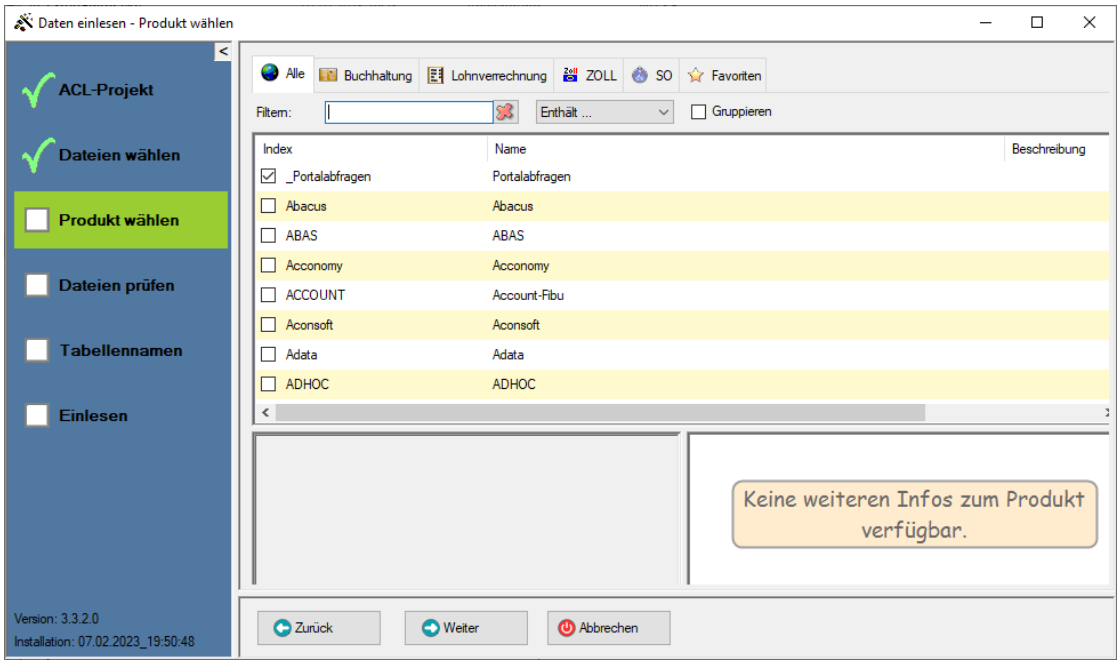


Abbildung 35: Datenimportassistent: Ansicht – Produkt auswählen

Dateien prüfen

Wenn die Daten für jedes Jahr exportiert wurden, müssen diese, sofern es nicht automatisch erkannt wird, einem bestimmten Jahr zugeordnet werden. Die Auswahl des Jahres erfolgt über ein Dropdown je Datei. In Abbildung 36 ist die Ansicht für das Dateien prüfen dargestellt. Kann eine Datei nicht dem ausgewählten Produkt zugeordnet werden, wird dieses durch ein rotes X gekennzeichnet.

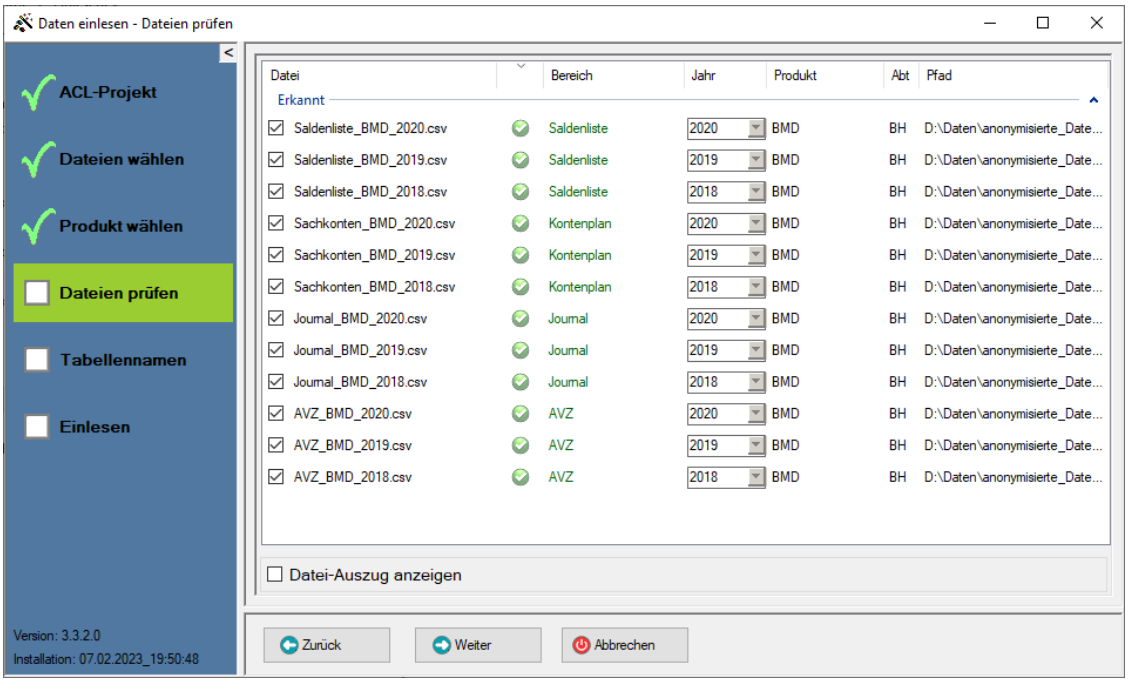


Abbildung 36: Datenimportassistent: Ansicht – Dateien prüfen

Tabellennamen

Nach dem Datenimport werden die Daten als Tabellen in der Software ACL Analytics angezeigt. Es wird für jede Tabelle automatisch ein Tabellenname vorgeschlagen. Der*die Anwender*in kann diese in diesem Schritt individuell anpassen. Dadurch soll auch das Überschreiben eventuell bestehender Tabellen vermieden werden. Abbildung 37 zeigt die Ansicht Tabellennamen. In dieser Ansicht mit roter Schrift und grauem Hintergrund dargestellte Bezeichnungen sind Tabellennamen, welche im Projekt bereits vorhanden sind. Durch Klick auf die Bezeichnungen können die Tabellennamen geändert werden oder diese Tabellen werden durch Betätigen der Schaltfläche „Weiter“ beim Datenimport überschrieben.

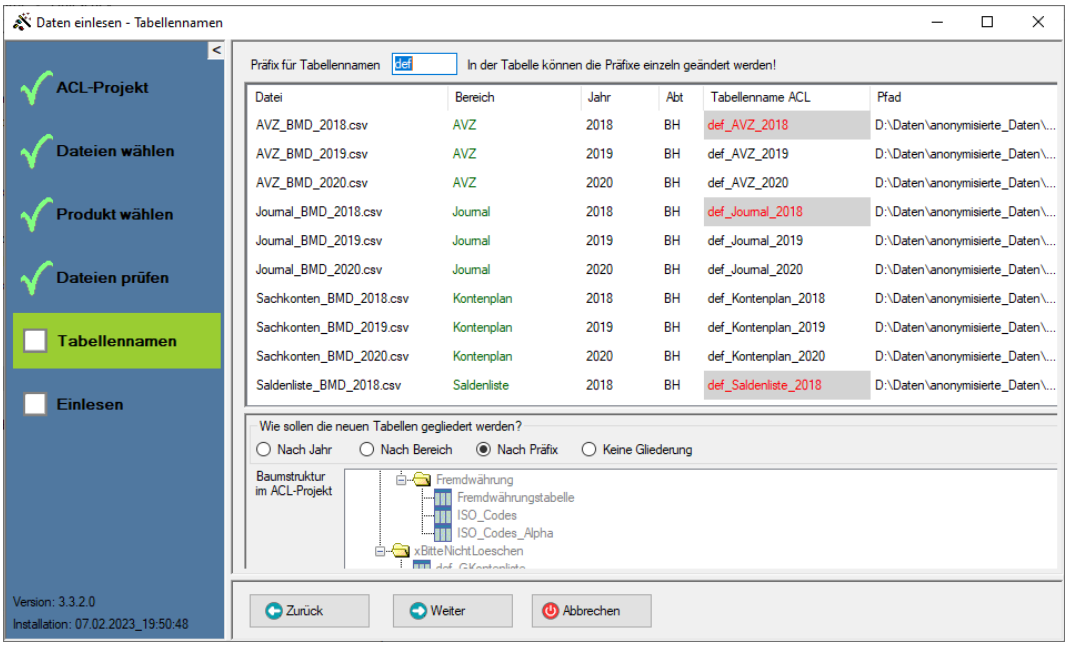


Abbildung 37: Datenimportassistent: Ansicht – Tabellennamen

Einlesen

In Abbildung 38 ist der letzte Schritt dargestellt. Hier kann noch die Option ausgewählt werden, ob eine Gesamtliste erstellt werden soll. Dabei werden alle Daten zu einer gesamten Tabelle zusammengeführt. Die Auswahl für die Gesamttabelle erfolgt über Checkboxen je Datei. Zusätzlich können hier auch automatische Datenanalysen gestartet werden. Durch Betätigen der Schaltfläche „Weiter“ wird der Importprozess gestartet und der Importassistent schließt sich selbstständig.

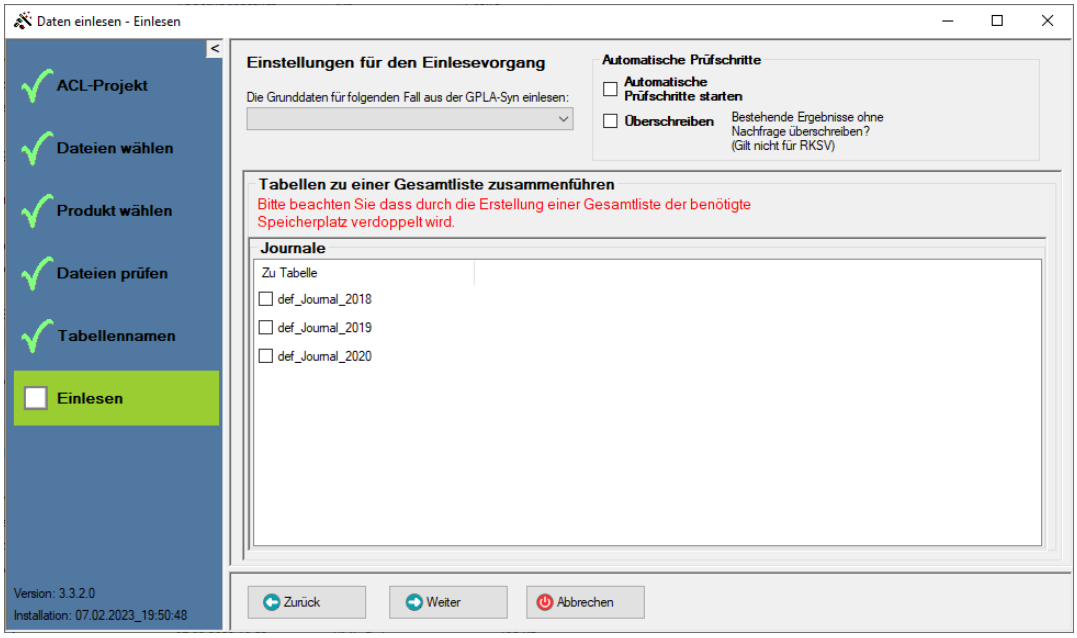


Abbildung 38: Datenimportassistent Ansicht – Einlesen

5.3 Beschreibung des Test-Cases

Der zu implementierende Test-Case ist der Datenimport mittels den im Abschnitt 5.2 beschriebenen Datenimportassistenten. Dieser Test-Case wird bereits manuell ohne Automatisierung durchgeführt. Beim manuellen Testen werden aktuell die Schritte einzeln ausgeführt und auf ein Fehlverhalten überprüft. Dieser Test-Case soll nun automatisiert durchgeführt werden. Dabei sollen über die Benutzeroberfläche alle Schritte, welche der*die Anwender*in beim Datenimport durchführen muss, durch den Softwareroboter durchgeführt werden.

Bei diesem Test-Case werden einzelne Daten aus dem Bereich Buchhaltung, Lohnverrechnung und Registrierkasse importiert. Dieser Test-Case entspricht dem am meisten durchgeführten Prozess, welchen der*die Anwender*in über den Datenimportassistenten durchführt.

Der Test-Case ist erfolgreich, wenn die Daten über den Datenimportassistenten ohne Fehler importiert werden können und am Ende alle Tabellen in der Software ACL Analytics vorhanden sind. Die Überprüfung, ob der Datenimportassistent richtig funktioniert, erfolgt einerseits dadurch, wenn es zu keinem Error während des Testdurchlaufes kommt und andererseits durch den Abgleich der Dateien, welche sich nach dem Import im jeweiligen ACL-Projekt befinden mit der Liste der Soll-Dateien. Fehlt eine Datei, war der Import nicht erfolgreich.

5.4 Beschreibung Softwareroboter für Test-Case

Dieser Abschnitt beschreibt die Skripte und die Abläufe der jeweiligen Softwareroboter. Zu Beginn werden das Skript und die Abläufe beim RPA-Tool OpenRPA und danach bei taskt beschrieben.

5.4.1 OpenRPA

In Abbildung 39 ist eine Übersicht bzw. die Struktur, bestehend aus den einzelnen Schritten des Softwareroboters, dargestellt. Diese Übersicht wurde mit einem Flowchart über die Entwicklungsumgebung des RPA-Tools OpenRPA erstellt. Hinter jedem dieser einzelnen Schritte verbirgt sich eine Sequenz an Tätigkeiten, die der Softwareroboter durchführen muss.

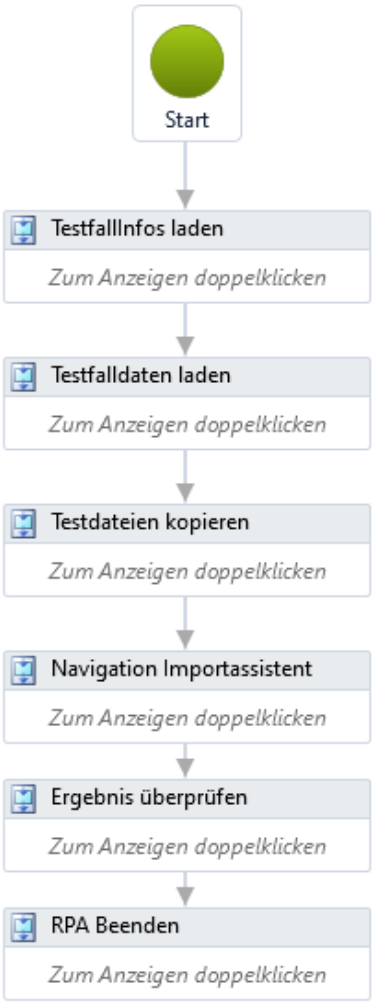


Abbildung 39: Die Schritte des Softwareroboters von OpenRPA beim Test-Case.

Die Schritte werden von oben nach unten abgearbeitet. Unmittelbar nach dem Start werden beim Schritt „Testfallinfos laden“ alle notwendigen Informationen aus der Test-Case XML vom Framework geladen. Danach werden im Schritt „Testfalldaten laden“ die Testfalldaten aus der Excel-Datei zum Test-Case vom Framework geladen und in Variablen vom Softwareroboter gespeichert. Im Anschluss erfolgt der Schritt „Testdateien kopieren“. Dabei werden die Testdaten aus den Test-Case-Ordnern in ein Verzeichnis des Testobjektes kopiert. Im Schritt „Navigation Importassistent“ wird der eigentliche Test durchgeführt. Dabei navigiert der Softwareroboter durch den Datenimportassistenten. Das Skript für den Softwareroboter in diesem Schritt befindet sich in Anhang F. Darin enthalten sind alle Tätigkeiten, die der Softwareroboter im Zuge des Test-Cases durchführen muss. Im Schritt „Ergebnis überprüfen“ prüft der Softwareroboter, ob der Datenimport erfolgreich war. Das wird Anhand eines Vergleiches der vorhandenen Dateien im ACL-Projektverzeichnis mit einer Sollliste an Dateien durchgeführt. Im letzten Schritt wird der Softwareroboter beendet.

5.4.2 Taskt

Bei der Umsetzung des Softwareroboters mit taskt wurde darauf geachtet, dass für einen besseren Vergleich die Struktur und die Abläufe gleich wie bei OpenRPA sind. In Abbildung 40 ist jenes Skript und dessen Schritte dargestellt, welches das Framework beim Arbeiten mit dem taskt aufruft. Hier sind die gleichen Schritte angeführt wie beim OpenRPA. Jeder Schritt ist wiederum ein eigenes Skript, welches die einzelnen Tätigkeiten des Softwareroboters enthalten.

Abbildung 40: Die Schritte des Softwareroboters von taskt beim Test-Case

Die Tätigkeiten, die der Softwareroboter bei den einzelnen Schritten ausführt, sind gleich wie bereits oben beim OpenRPA beschrieben. Das Skript für die Navigation durch den Datenimportassistenten ist in Anhang G dargestellt. Auch hier wiederum ist dieses Skript jenes, welches dazu verwendet wird, um den eigentlichen Test durchzuführen. Alle anderen Skripte sind nur vorbereitende oder nachbereitende Tätigkeiten, die der Softwareroboter ausführen muss.

5.5 Praktische Anwendung des Frameworks

Das Framework, wie auch die oben beschriebenen Test-Cases, wurde im Entwicklungsteam implementiert. Dadurch steht diese Testautomatisierung mittels RPA dem gesamten Team während der Entwicklung und auch zum Testen vor einem Release zur Verfügung. Die Testfälle wurden auf einem Netzlaufwerk abgelegt. Dadurch hatten alle im Team denselben Stand bei den Testfällen. Alle Anpassungen konnten so zentral durchgeführt werden.

Die praktische Anwendung des Frameworks erfolgte in mehreren Entwicklungsphasen. Wird der Datenimportassistent weiterentwickelt, wird die Testautomatisierung bereits bei der Entwicklung verwendet, um zu prüfen, ob die Entwicklung eventuelle Regressionen zu Folge hat. Da diese Tests durch die Testautomatisierung schnell durchgeführt werden können, werden diese Tests auch öfters durchgeführt, wodurch Regressionen schneller erkannt werden. In der Entwicklungsphase war es zudem auch

wichtig, dass bei Entwicklungen, welche Auswirkungen auf die Testautomatisierung haben, auch sofort beim entsprechenden Softwareroboter geändert wurden – zum Beispiel, wenn bewusst die Ansicht bei der Benutzeroberfläche geändert wurde oder im Prozess neue Steuerelemente implementiert werden.

Zudem wird die Testautomatisierung dazu verwendet, um vor einem Release eine abschließende Qualitätskontrolle durchzuführen. Das Test-Reporting wird als Nachweis der Qualitätssicherung entsprechend archiviert und kann so zu einem späteren Zeitpunkt für Vergleiche oder bei Problemen als Nachweis für den Umfang und für die erzielten Ergebnisse bei der Qualitätskontrolle herangezogen werden.

5.6 Ergebnisse der Evaluierung

Mit der Implementierung von Test-Cases in den Prototyp und der Anwendung des Frameworks in der Praxis wurde die Funktionsweise des Frameworks evaluiert. Bei der Evaluierung der Funktionsweise wurde darauf geachtet, dass das konzipierte Framework für die Testautomatisierung geeignet ist und es zu keinem technischen Fehler bei der Anwendung des Frameworks kommt. Bei der praktischen Anwendung wurde evaluiert, ob das Framework mit den Prozessen der Testautomatisierung eingesetzt werden kann.

5.6.1 Anwendung in der Praxis

Es hat sich gezeigt, dass der Einsatz des Frameworks ohne Programmierkenntnisse möglich ist. Innerhalb eines Testteams bedarf es nur RPA-Entwickler*innen, welche bei der von der RPA-Software mitgelieferten Entwicklungsumgebung des Softwareroboters für die jeweiligen Test-Cases entwickeln. Sobald die Softwareroboter in das Framework integriert sind, können die Tests von Entwickler*innen oder von Fachexpertinnen und Fachexperten ausgeführt werden. Dadurch, dass die Test-Cases für zwei unterschiedliche RPA-Tools umgesetzt und in der Praxis angewendet wurden, wurde gezeigt, dass das Framework problemlos im Zusammenhang mit unterschiedlichster RPA-Software verwendet werden kann. Das wurde dadurch erreicht, weil die Entwicklung der Softwareroboter auch weiterhin in den jeweiligen Entwicklungsumgebungen der jeweiligen RPA-Software stattfindet. Zudem hat die praktische Anwendung des Frameworks gezeigt, dass Test-Cases mit dem im Abschnitt 2.1.1 beschriebenen Testprozess umgesetzt werden können. Dadurch können bereits etablierte Prozesse auch mit dem Framework beibehalten werden.

5.6.2 Steuerung der Softwareroboter

Die Funktionsweise für die Steuerung der Roboter hat einwandfrei funktioniert. Da bei mehreren Testfällen die einzelnen Softwareroboter sequenziell aus dem Framework heraus gestartet werden, bleibt die Kontrolle für jeden einzelnen Testfall beim Framework. Dadurch kann bei einem Fehler der einzelne Softwareroboter beendet und der nächste problemlos gestartet werden. Abhängigkeiten zwischen einzelnen Testfällen waren kein Problem, da die Anwender*innen durch die Reihenfolge, wie die einzelnen Testfälle ausgewählt wurden, selbst entscheiden konnten, wie die Reihenfolge beim Ausführen ist. Zudem hat sich gezeigt, dass die Anzeige der ausgewählten Testfälle vor dem Testdurchlauf für die Anwender*innen von großem Wert war, da hier durch die gute Übersicht bereits vor dem Testdurchlauf festgestellt werden konnte, ob die richtigen Testfälle ausgewählt und die Reihenfolge stimmt.

5.6.3 Erweiterung der RPA-Software

Die Implementierung der erforderlichen Erweiterungen durch DLL in die jeweiligen RPA-Tools war problemlos möglich. Die darin enthaltenen Funktionen konnten über die Entwicklungsumgebung der RPA-Software in die Softwareroboter integriert werden. Die Verwendung der Funktionen durch den Softwareroboter hat ohne Probleme funktioniert und der bereitgestellte Funktionsumfang war für die Testautomatisierung ausreichend.

5.6.4 Verwaltung der Testdaten

Durch die Verwaltung der Testdaten durch das Framework in einer eigenen Verzeichnisstruktur wird der Zugriff auf diese durch das Framework und den Softwareroboter erleichtert. Neben einer lokalen Ablage ist auch die Ablage auf einem Netzlaufwerk möglich, da beim Framework der Pfad zu diesem Verzeichnis individuell angepasst werden kann. Vor allem bei der praktischen Anwendung des Frameworks hat sich gezeigt, dass die zentrale Ablage und Verwaltung der Testfälle von Vorteil sind. Durch den Zugriff aus dem Framework heraus auf diese Daten und Dateien können sich die Anwender*innen schnell einen Überblick über diese Testdaten verschaffen. Zudem können Änderungen oder Erweiterungen der Testdaten rasch durchgeführt werden. Durch die Anzeige der Beschreibung für jeden einzelnen Testfall im Framework können komplizierte Testfälle ausführlich beschrieben werden. Dadurch wird ein besseres Verständnis für den Testfall vermittelt und bei einem eventuell auftretenden Fehlverhalten des Testobjektes kann der*die Anwender*in schnell die Ursache feststellen.

5.6.5 Test-Reporting

Durch die Erstellung des Test-Reports in HTML können die Ergebnisse grafisch einfach aufbereitet werden. Zudem können auch Funktionen für die Interaktion der Anwenderin bzw. des Anwenders bereitgestellt werden. Das Dokument kann beispielsweise auch gedruckt, gespeichert oder versendet werden, was zusätzliche Vorteile bringt. Die Trennung der Bereiche Testergebnisse und Fortschrittsprotokollierung hat in der praktischen Anwendung gezeigt, dass sich dadurch ein besserer Überblick verschafft werden kann. Die Schritte für die Fortschrittsprotokollierung können die Entwickler*innen der Softwareroboter für den jeweiligen Testfall selbst festlegen. Bereits bei der Anpassung und Stabilisierung der Softwareroboter zeigt sich, ob die Protokollierung ausreichend vorhanden ist. Die praktische Anwendung hat gezeigt, dass eine sehr detaillierte Protokollierung bei fast nahezu jedem Schritt, den der Softwareroboter macht, das Auffinden der Fehler und die möglichen Ursachen enorm erleichtert. Die Übersicht beim Test-Reporting machte es möglich, auf einen Blick zu erkennen, wie das Ergebnis beim gesamten Testdurchlauf ausgefallen ist. Das Einbeziehen historischer Ergebnisse erleichtert die Dokumentation von Veränderungen. Die Archivierung des Test-Reportings zur Dokumentation der Qualitätssicherung ist einfach möglich. Zudem kann jeder außerhalb des Frameworks das Test-Reporting in einem Browser öffnen. Das Hinzufügen von Anmerkungen zum Testdurchlauf und von den Systeminformationen hat sich in der Praxis als sehr nützlich erwiesen. Testdurchläufe konnten beschrieben werden und Informationen zur Version des Testobjektes konnten ergänzt und in weiterer Folge mitdokumentiert werden.

Die Ergebnisse der Evaluierung zeigen, dass die Anforderung von Kapitel 3.2 vollständig umgesetzt wurde. Das Framework wurde so konzipiert und entwickelt, dass die Prozesse bei der Umsetzung der Testautomatisierung gleichbleiben können. Daher ist es möglich, mit einem Framework, welches so konzipiert ist, wie auf Seite 33 in Abbildung 9: Architektur des Frameworks für die Testautomatisierung mit RPA (eigene Darstellung) dargestellt, die Testautomatisierung mittels RPA durchzuführen.

5.7 Vor- und Nachteile des Frameworks

Ein Vorteil des Frameworks ist die leichte Bedienung. Die Benutzeroberfläche ist einfach strukturiert. Das Test-Reporting bietet einen guten Überblick und mit den Details zu den einzelnen Testfällen können schnell Fehlerursachen erkannt und nachvollzogen werden.

Dadurch, dass das Test-Reporting in HTML-Dateien gespeichert wird, können diese einfach archiviert und auch schnell außerhalb des Frameworks in jedem Browser angezeigt werden. Das erleichtert die Zusammenarbeit mit externen Partnern, da das Test-Reporting einfach versendet und so weiteren Personen mit allen Informationen und Funktionalitäten zur Verfügung steht.

Fertige Softwareroboter können ohne Programmieraufwand in das Framework integriert werden. Die dazugehörigen Testdaten können strukturiert in die Verzeichnisstruktur, welches das Framework vorgibt, abgelegt werden. Durch die Verwendung von Excel ist eine einfache Bedienung und die schnelle Durchführung von Änderungen möglich.

Die Prozesse, welche für die Testautomatisierung zur Anwendung kommen, bleiben auch bei diesem Framework unverändert. Dadurch müssen bereits erfahrene Tester*innen keine zusätzlichen Prozesse oder Abläufe trainieren. Zudem kann auch bestehende Literatur über die Testautomatisierung herangezogen werden, um sich die Grundkenntnisse anzueignen und danach die Testautomatisierung mit diesem Framework im Zusammenhang mit RPA-Software umzusetzen.

Durch die Auswahl der einzelnen Testfälle können alle auf Seite 18 in Abbildung 5 beschriebenen Techniken von Regressionstests mit dem Framework durchgeführt werden. Dadurch ist eine Anpassung der Tests auf die jeweiligen Rahmenbedingungen möglich.

Das Framework ist an keine spezielle RPA-Software gebunden. Vor der Umsetzung der Testautomatisierung kann eine geeignete RPA-Software ausgewählt und in das Framework integriert werden.

Ein Nachteil ist, wenn bereits mehrere Testfälle im Framework implementiert wurden, ist man an die jeweilige RPA-Software gebunden. Möchte man diese ändern, müssen auch alle Softwareroboter geändert werden.

Ein weiterer Nachteil des Frameworks ist, dass das Testobjekt und das Framework wie auch die RPA-Software in derselben Umgebung installiert sein müssen. Was zufolge hat, dass entweder alles in einer virtuellen Maschine betrieben werden muss oder dass man das Risiko einer Beeinträchtigung der Funktionsfähigkeit der RPA-Software durch eine eventuelle Fehlfunktion in Kauf nimmt.

5.8 Zusammenfassung

Die Evaluierung erfolgte auf zwei Wegen. Einerseits wurden Test-Cases mit dem Framework implementiert. Das dient vor allem der Evaluierung der Funktionsweise und ob das konzipierte Framework für die Testautomatisierung geeignet ist. Andererseits wurde das Framework auch in der praktischen Anwendung getestet. Für die Entwicklung der Softwareroboter für die einzelnen Test-Cases war eine eigene Vorgehensweise erforderlich. Nur durch eine iterative Vorgehensweise, bei der die Softwareroboter Schritt für Schritt verbessert wurden, konnten die Softwareroboter eine entsprechende Stabilität aufweisen, sodass sie auch für die Testautomatisierung eingesetzt werden konnten. Die Test-Cases wurden auf ein Testobjekt aus dem beruflichen Umfeld angewendet. Dabei handelt es sich um einen selbst entwickelten Datenimportassistenten für die Software ACL Analytics. Die ausgewählten Test-Cases entsprechen jenen Prozessen, die am häufigsten von den Anwendenden durchgeführt werden. Dabei handelt es sich um den Datenimport von ausgewählten Buchhaltungs-, Lohnverrechnungs- und Registrierkassendaten. Da das Framework nicht an eine RPA-Software gebunden sein soll, wurden die Test-Cases mit zwei unterschiedlichen RPA-Tools umgesetzt. Dabei wurde darauf geachtet, dass für eine bessere Vergleichbarkeit die grundsätzliche Vorgehensweise der Softwareroboter dieselbe ist. Das Framework mit den implementierten Test-Cases wurde auch in der Praxis angewendet. Zu diesem Zweck wurde die Testautomatisierung in zwei unterschiedlichen Entwicklungsphasen eingesetzt. Einerseits während der Entwicklung, um schnell Regressionen erkennen zu können. Andererseits am Ende, um abschließend feststellen zu können, ob das Testobjekt richtig funktioniert bzw. auch als Nachweis der Qualitätssicherung. Die Evaluierungsergebnisse zeigten auch, dass die Anforderungen umgesetzt wurden und dass das darauf aufbauende Framework für die Umsetzung der Testautomatisierung mittels RPA geeignet ist.

6 Beantwortung der Forschungsfrage, Zusammenfassung und Ausblick

In diesem Kapitel wird die Forschungsfrage beantwortet und die in der vorliegenden Arbeit erzielten Erkenntnisse zusammengefasst. Am Ende dieses Kapitels erfolgt der Ausblick über die Diffusion des Frameworks durch Implementierung im Unternehmen und als Open-Source-Projekt auf GitHub.

6.1 Beantwortung der Forschungsfrage

Durch die Evaluierung des Frameworks wurde gezeigt, dass die Architektur des Frameworks, so wie auf Seite 33 in Abbildung 9 dargestellt, für die Testautomatisierung mittels RPA-Software herangezogen werden kann. Daher kann die Forschungsfrage, *„Wie muss ein Framework für den Einsatz von Robotic Process Automation Tools zur Testautomatisierung bei Regressionstests im End-to-End-Bereich konzipiert sein, damit die Steuerung der einzelnen Test-Cases in Form von RPA-Prozessen sowie ein Test-Reporting der Ergebnisse ermöglicht wird?“* anhand der gestellten Anforderungen und der daraus abgeleiteten Architektur beantwortet werden. Ein Framework für die Testautomatisierung mittels RPA-Tools muss so gestaltet sein, dass es die im Abschnitt 3.2.2 beschriebenen allgemeinen Anforderungen an die Testautomatisierung erfüllt. Dazu gehört, dass die Testautomatisierung anhand des im Kapitel 2.1.1 beschriebenen Testprozesses umgesetzt werden kann.

Ein Framework muss auch die Verwaltung der Testdaten ermöglichen. Da es sich hier um ein Framework handelt, muss die Verwendung von unterschiedlicher RPA-Software ermöglicht werden. Es darf keine Bindung an ein spezielles Produkt geben. Für Regressionstests muss es möglich sein, den Testdurchlauf individuell anpassen zu können. Dazu ist es erforderlich, dass die einzelnen Test-Cases individuell durch die Anwendenden ausgewählt und beim Testdurchlauf ausgeführt werden. Da es bei der Testautomatisierung zu einer Vielzahl an einzelnen Test-Cases kommen kann, müssen diese für eine bessere Übersicht und Verständnis einzeln beschrieben werden. Die Beschreibung macht es auch möglich, dass Fachexpertinnen und Fachexperten, welche nicht selbst den Test-Case entwickelt haben, sich schnell zurechtfinden und auch die Tests durchführen können. Bei der Konzeption müssen auch die im Abschnitt 3.2.2 beschriebenen Anforderungen an die Steuerung der Softwareroboter berücksichtigt werden. Dazu zählen, dass mehrere Testfälle von den Anwendenden ausgewählt und in

einer individuellen Reihenfolge abgearbeitet werden müssen. Zudem muss die Abarbeitung der Testfälle durch je einen einzelnen Softwareroboter in einer sequenziellen Reihenfolge erfolgen. Dadurch können die Softwareroboter und deren Fortschritt überwacht und bei Problemen beendet und mit dem nächsten fortgesetzt werden. Bei der Konzeption des Frameworks wurde festgestellt, dass bei der jeweiligen RPA-Software Anpassungen erforderlich sind, damit der jeweilige Softwareroboter Informationen zwischen ihm und dem Framework austauschen kann. Auf Basis der Erkenntnisse dieser Arbeit hat sich herausgestellt, dass diese Anpassungen in Form von DLL am besten geeignet sind. Jede hier recherchierte RPA-Software kann ihre Funktionen mittels DLL erweitern. Beim Einbinden dieser DLL gibt es jedoch Unterschiede. Manche RPA-Software kann die jeweiligen Funktionen direkt aus der DLL heraus aufrufen und bei anderen müssen diese als Aktivität eingebunden werden, damit diese bei der Entwicklung der Softwareroboter eingebunden werden können. Diese Funktionen müssen vom Framework zur Verfügung gestellt und bereits bei der Konzeption berücksichtigt werden.

Mit den implementierten Funktionen in den Softwarerobotern ist es möglich, dass das Framework beim Start des Softwareroboters diesen mit entsprechenden Informationen über den Testfall liefert, das Framework den Fortschritt des Softwareroboters überwacht und der Softwareroboter entsprechende Fortschrittsinformationen und Testergebnisse an das Framework zurück liefert. Diese Informationen werden vom Framework benötigt, um die im Abschnitt 3.2.3 beschriebenen Anforderungen an das Test-Reporting zu erfüllen. Zu diesen Anforderungen gehört, dass jeder Testdurchlauf durch den*die Tester*in beschrieben können werden muss. Vor allem bei der Dokumentation der Qualitätssicherung oder bei einem späteren Vergleich der Ergebnisse ist die Beschreibung sehr wertvoll. Dazu muss diese Beschreibung in das Test-Reporting übernommen werden. Bei der Konzeption des Test-Reportings wurden zwei Bereiche, die Übersicht und Details, festgelegt. Die Ergebnisse der einzelnen Test-Cases werden zu einer Übersicht zusammengefasst. Die Übersicht soll das Ergebnis des Testdurchlaufes darstellen. Zu einer besseren Übersicht werden diese auch grafisch aufbereitet. Die Softwareroboter liefern zu jedem Test-Case die Information OK, FAIL oder Error. OK und FAIL dienen dem Ergebnis eines Soll- und Ist-Vergleiches. Der Error wird entweder vom Softwareroboter selbst gesetzt oder durch das Framework, wenn es zu Problemen mit dem Softwareroboter kommt.

Im zweiten Bereich des Test-Reportings werden die Details mit dem Ergebnis- und dem Fortschrittsprotokoll angeführt. Bei einem Test-Case kann es zu mehreren Soll-Ist-

Vergleichen kommen. Diese werden einzeln im Ergebnisprotokoll aufgelistet. Im Fortschrittsprotokoll werden die einzelnen Fortschritte, die der Softwareroboter bei dem jeweiligen Test-Case erreicht hat, aufgelistet. Das sind jene Schritte, die bei der Entwicklung des Softwareroboters durch den*der jeweiligen Entwickler*in individuell festgelegt wurden. Dadurch wird das Auffinden etwaiger Fehler leichter gemacht. Bei der Konzeption des Frameworks wurde das Test-Reporting in Form von HTML umgesetzt. Dadurch kann das Test-Reporting interaktiv gestaltet werden und die einzelnen HTML-Dateien können verschoben, versendet oder woanders für die Archivierung abgelegt werden. Bei der Konzeption wurde darauf geachtet, dass die Ergebnisse auch mit früheren Ergebnissen automatisch verglichen werden können. Die aus den Anforderungen heraus konzipierte Architektur des Frameworks ist auf Seite 33 in Abbildung 9 dargestellt. Diese zeigt schematisch, wie die RPA-Software in das Framework integriert werden kann. Der im Zuge dieser Arbeit umgesetzte Prototyp, welcher auf Basis dieser Architektur entwickelt wurde, hat gezeigt, dass mit einem so konzipierten Framework eine Testautomatisierung mit RPA möglich ist und dabei den Tester*innen ein brauchbares Ergebnis zur Verfügung gestellt werden kann.

6.2 Zusammenfassung

Mit dieser Arbeit wurde das Ziel verfolgt, herauszufinden, wie mit RPA eine geeignete Testautomatisierung in Bezug auf die Steuerung der einzelnen Test-Cases sowie für das Test-Reporting möglich ist. Zu diesem Zweck wurde ein prototypisches Framework entwickelt. Um dieses entwickeln bzw. um die Forschungsfrage beantworten zu können, wurde der methodische Rahmen des Design Science Research von Hevner et al. angewendet. Dieser methodische Rahmen ermöglicht es, durch ein strukturiertes Vorgehen das prototypische Framework zu entwickeln. Dabei wurden die vier Iterationsschritte Analyse, Entwurf, Evaluation und Diffusion nach Österle et al. angewendet. Die Analyse wurde im Kapitel 2, der Entwurf im Kapitel 3, die Evaluation in den Kapiteln 4 und 5 und die Diffusion am Ende dieses Kapitels behandelt. Bei der Analyse wurden die Grundlagen und der Stand der Technik recherchiert. Dabei wurden die grundlegenden Prozesse bei der Testautomatisierung von der Testplanung über die Testrealisierung bis hin zum Abschluss der Testaktivität beschrieben. Bei der Konzeption des Frameworks wurde darauf geachtet, dass auch die bereits etablierten Prozesse der Testautomatisierung angewendet werden können. Die unterschiedlichen Teststufen sowie die Regressionstests wurden bei der Analyse beschrieben. Die Testautomatisierung mit dem RPA sollte im End-to-End-Bereich und bei

Regressionstests angewendet werden. Dabei zeigte sich, dass vor allem bei Regressionstests eine Auswahl einzelner Testfälle möglich sein muss.

Die Auswahl soll je nach Priorität oder vorhandenen Ressourcen durch den Anwendenden erfolgen. Im End-to-End-Bereich, welcher auch dem Systemtest zuzuordnen ist, werden ganze Use-Cases oder ausgewählte Abläufe als Test-Cases für die Testautomatisierung herangezogen. Dabei geht es um die Nachahmung von Abläufen, welche die Anwendenden durchführen. Diese Art von Tests ist ideal für RPA-Tools, da diese auf demselben Prinzip beruhen. Softwareroboter werden dort eingesetzt, wo gleichbleibende strukturierte Aufgaben und Tätigkeiten auszuführen sind und sollen dabei den Menschen von gleichbleibenden monotonen Aufgaben befreien. Dabei interagiert der Softwareroboter wie auch der Mensch über die Benutzeroberfläche mit dem jeweiligen IT-System. Die Softwareroboter können bei der Interaktion mit dem IT-System die einzelnen Steuerelemente erkennen und betätigen. Dadurch ist ein regelbasiertes Ausführen von Funktionen in den Anwendungssystemen möglich.

Die Entwicklung solcher Softwareroboter bedarf keiner ausführlichen Programmierkenntnis, da diese entweder mittels Drag-and-Drop-Funktionen einen Workflow-orientierten Ansatz oder einen Skript-orientierten Ansatz verfolgt. Da das Framework mit mehreren unterschiedlichen RPA-Tools die Testautomatisierung durchführen und somit nicht an eine spezielle Software gebunden sein soll, mussten auch die Eigenschaften mehrerer am Markt vorhandener RPA-Tools betrachtet werden. Insgesamt wurden 4 RPA-Tools und deren Eigenschaften analysiert. Dabei handelt es sich um zwei Lizenzprodukte und zwei Open-Source-Lösungen. Bei der Betrachtung ging es primär darum, wie kann durch das Framework die Steuerung und die Überwachung der Softwareroboter ermöglicht werden und welche Schnittstellen gibt es dafür. Als Ergebnis wurde festgestellt, dass alle RPA-Tools durch DLL erweitert und mittels Command-Line gestartet werden können.

Aufbauend auf diesen Ergebnissen wurde mit dem nächsten Iterationsschritt der Entwurf gestartet. Bei diesem Schritt wurde das Framework konzipiert. Ziel dieser Konzipierung war, darauf aufbauend einen funktionalen Prototyp entwickeln zu können. Dabei lag der Fokus auf der Beantwortung der Forschungsfrage. Zu Beginn wurden auf Basis der Analyse die Anforderungen an das Framework zusammengefasst. Diese gliederten sich in grundlegende Anforderungen, in Anforderungen an die Steuerung der Softwareroboter und in Anforderungen an das Test-Reporting. Darauf aufbauend wurde eine Gesamtarchitektur des Frameworks skizziert. Aus dieser Architektur geht hervor,

dass das Framework die Verwaltung der Testfälle übernehmen muss. Dazu gehört eine entsprechende Verzeichnisstruktur, in der die Testfälle mit der Beschreibung, den Testdaten und eventuell weiterer erforderlicher Dateien abgelegt werden. Die einzelnen Softwareroboter werden sequenziell einzeln gestartet und arbeiten so die einzelnen Testfälle ab. Diese Softwareroboter liefern dabei Fortschritts- und Testergebnisse. Am Ende eines Testdurchlaufes wird im Framework ein entsprechendes Test-Reporting mit den Ergebnissen des Testdurchlaufes angezeigt.

Nach der Fertigstellung des Konzeptes erfolgte der nächste Iterationsschritt, die Evaluation. Bei der Evaluation wurde zu Beginn der Prototyp umgesetzt. Der Prototyp wurde in der Programmiersprache C# geschrieben und bekam eine Benutzeroberfläche, über die der*die Anwendende die einzelnen Testfälle bearbeiten und auswählen kann. Das Framework war so gestaltet, dass der*die Anwender*in ohne Programmierkenntnisse die einzelnen Softwareroboter implementieren kann. Der*die Anwender*in kann die einzelnen Testfälle für einen Testdurchlauf auswählen. Jeder Testfall wie auch der Testdurchlauf können entsprechend beschrieben werden. Dadurch kann einerseits jedem Anwendenden ein schneller Überblick über die Testfälle und deren Abläufe gegeben werden und andererseits kann bei einem eventuell auftretenden Fehler schneller nachvollzogen werden, worin das Problem liegt.

Die Beschreibung des Testdurchlaufes ermöglicht es, dass eine entsprechende Dokumentation über den Zweck und in weiterer Folge für das Qualitätsmanagement durchgeführt werden kann. Diese Beschreibung wird auch in das Test-Reporting übernommen. Die Entwicklung des Test-Reportings wurde mit HTML durchgeführt. Dadurch ist eine grafische Aufbereitung der Ergebnisse, aber auch eine Interaktion über Links möglich.

Das Test-Reporting gliedert sich in 2 Bereiche, eine Übersicht und eine Detailansicht. Neben dem Framework mussten auch entsprechende Erweiterungen der Funktionen von RPA-Tools entwickelt werden. Mit diesen Funktionen ist es RPA-Tools möglich, diese für die Testautomatisierung einzusetzen. Dabei geht es um die Möglichkeiten, entsprechende Fortschrittsinformationen und Testergebnisse zurück an das Framework liefern zu können. Die Funktionen wurden durch entsprechende DLLs bereitgestellt, da bei der Analyse festgestellt wurde, dass alle RPA-Tools durch DLLs die Funktionen erweitern können. Nach der Fertigstellung des Prototyps erfolgte die eigentliche Evaluierung durch die Implementierung von Test-Cases und Anwendung in der Praxis.

Bei der Evaluierung wurden mehrere Testfälle ausgewählt, welche für die Testautomatisierung mittels des Frameworks umgesetzt werden sollten. Dazu wurden die jeweiligen Softwareroboter entwickelt und die Testfälle in das Framework integriert. Um festzustellen, ob das Framework auch von mehreren RPA-Tools eingesetzt werden kann, wurden die Softwareroboter mit OpenRPA und taskt umgesetzt und implementiert. Bei der Entwicklung der Softwareroboter wurden durch iteratives Vorgehen diese verbessert und stabilisiert. Dadurch konnte verhindert werden, dass es zu einem Fehlverhalten der Softwareroboter und daraus resultierend zu einem falschen Testergebnis kommt. Die so entwickelten Testfälle wurden in der Praxis für die Testautomatisierung angewendet. Die praktische Anwendung hat gezeigt, dass mit dem Framework die Testautomatisierung mittels RPA gut funktioniert. Die Anforderungen wurden umgesetzt und das darauf aufbauende konzipierte Framework ist für die Testautomatisierung geeignet.

6.3 Ausblick über die Diffusion des Frameworks

In diesem Abschnitt erfolgt die Beschreibung der Diffusion des Frameworks. Die Diffusion wird in zwei Bereiche unterteilt. Einerseits wird die Implementierung des Frameworks im Unternehmen und andererseits die Veröffentlichung des Frameworks als Open Source Software im Internet beschrieben.

6.3.1 Implementierung des Frameworks im Unternehmen

Das im Zuge dieser Arbeit entwickelte Framework soll im Unternehmen für die Testautomatisierung zum Einsatz kommen. Für diesen Zweck soll der Prototyp weiterentwickelt werden, sodass am Ende eine voll funktionsfähige Anwendung zur Verfügung steht. Zudem müssen im Unternehmen auch organisatorische Maßnahmen durchgeführt werden, damit die Testautomatisierung umgesetzt werden kann. Zu den erforderlichen Maßnahmen gehören eine entsprechende Schulung der Mitarbeiter*innen und die Integration der Testautomatisierung in die Prozesse der Entwicklung und Qualitätssicherung. In weiterer Folge müssen alle im Team mit den Prozessen Testautomatisierung vertraut sein, da die Testautomatisierung einen kontinuierlichen Prozess bei der Softwareentwicklung darstellt.

6.3.2 Veröffentlichung des Frameworks als Open-Source-Projekt

Damit das Framework mehreren Unternehmen bzw. Softwareentwicklerinnen und Softwareentwicklern zur Verfügung steht, soll dieses auch über GitHub zur Verfügung

gestellt werden. GitHub ist ein webbasierter Dienst, welcher Unternehmen oder einzelnen Entwicklerinnen und Entwicklern das Speichern, Verwalten und das Nachvollziehen von Änderungen am Code ermöglicht. Für ein Public Code Repository steht einem GitHub kostenlos zur Verfügung. Dadurch ist GitHub besonders bei Open-Source-Projekten beliebt [Gith00]. Über ein entsprechendes Public Code Repository bei GitHub kann der Sourcecode vom Framework veröffentlicht werden. Dieses Repository kann sich jeder klonen und entsprechend verwenden oder sogar für eigene Zwecke weiterentwickeln. Zudem könnten andere Entwickler*innen an der Weiterentwicklung mithelfen. Sie können Änderungen vornehmen und über Pull Requests können diese öffentlich diskutiert und in weiter Folge auch in den Sourcecode übernommen werden. Dadurch könnte das Framework durch eine breite Community von Softwareentwicklerinnen und Softwareentwicklern weiterentwickelt und verbessert werden.

Literaturverzeichnis

[AABK05] AMIRY, SAMIR; ARMBRUST, OVE; BERGER, JULIA; KLINCK, JANINE; LUTTENBERGER, KONSTANTIN: Research Lab Rheinland-Pfalz Testen und Testautomatisierung: Anforderungen an Testwerkzeuge und Marktstudie. In: *IESE-Report Nr* Bd. 131 (2005)

[ArHH21] ARNAUTOVIC, HANKA; HABEGGER, ANJA; HALLER, STEPHAN: Grundlagen der Robotic Process Automation. In: *Digital Business*, Springer Gabler, Wiesbaden (2021), S. 57–83

[ArOS04] ARMBRUST, OVE; OCHS, MICHAEL; SNOEK, BJÖRN: Stand der Forschung von Software-Tests und deren Automatisierung.

[Auto00] AUTOMATION ANYWHERE: *Globaler Marktführer für intelligente Automatisierung & RPA*. URL https://www.automationanywhere.com/de/. – abgerufen am 2022-08-07. – Automation Anywhere

[BBSG15] BUCSICS, THOMAS; BAUMGARTNER, MANFRED; SEIDL, RICHARD; GWIHS, STEFAN: *Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken*: dpunkt.verlag, 2015 – ISBN 978-3-86490-194-2

[BeKS20] BENNER-WICKNER, MARIAN; KNEUPER, RALF; SCHLÖMER, INGA: *Leitfaden für die Nutzung von Design Science Research in Abschlussarbeiten*: IUBH Discussion Papers – IT & Engineering, 2020

[BFGL18] BAYERL, EBERHARD; FABIAN, ERNST; GINNER, MICHAEL; LEZUO, DAVIC: Fleißige Roboter. In: *KPMG Österreich*, *Dimensionen*. (2018), Nr. April 2018, S. 20–21

[Bill22] WAGNER, BILL: *Überblick über C# – Übersicht*. URL https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/. – abgerufen am 2022-04-14

[Bret20] BRETTSCHNEIDER, JENNIFER: Bewertung der Einsatzpotenziale und Risiken von Robotic Process Automation. In: *HMD Praxis der Wirtschaftsinformatik* Bd. 57 (2020), Nr. 6, S. 1097–1110. – Company: SpringerDistributor: SpringerInstitution: SpringerLabel: Springernumber: 6publisher: Springer Fachmedien Wiesbaden

[CeSS20] CERNAT, MARINA; STAICU, ADELINA-NICOLETA; STEFANESCU, ALIN: Improving UI Test Automation using Robotic Process Automation. In: *ICSOFT*, 2020, S. 260–267

[Char00] *Chart.js | Open source HTML5 Charts for your website*. URL https://www.chartjs.org/. – abgerufen am 2022-08-20

[CzAu18] CZARNECKI, CHRISTIAN; AUTH, GUNNAR: Prozessdigitalisierung durch Robotic Process Automation. In: BARTON, T.; MÜLLER, C.; SEEL, C. (Hrsg.): *Digitalisierung in Unternehmen: Von den theoretischen Ansätzen zur praktischen Umsetzung*, *Angewandte Wirtschaftsinformatik*. Wiesbaden: Springer Fachmedien, 2018 – ISBN 978-3-658-22773-9, S. 113–131

[CzBA19] CZARNECKI, CHRISTIAN; BENSBERG, FRANK; AUTH, GUNNAR: Die Rolle von Softwarerobotern für die zukünftige Arbeitswelt. In: *HMD Praxis der Wirtschaftsinformatik* Bd. 56 (2019), Nr. 4, S. 795–808. – Company: SpringerDistributor: SpringerInstitution: SpringerLabel: Springernumber: 4publisher: Springer Fachmedien Wiesbaden

[Dela00] DELAND-HAN: *Dynamic Link Library (DLL) – Windows Client*. URL https://docs.microsoft.com/de-de/troubleshoot/windows-client/deployment/dynamic-link-library. – abgerufen am 2022-08-07

[Diet02] DIETZSCH, ANDREAS: Frameworks. In: *Systematische Wiederverwendung in der Software-Entwicklung*. Wiesbaden: Deutscher Universitätsverlag, 2002 – ISBN 978-3-663-11580-9, S. 77–89

[Docu00] *Documentation for Visual Studio Code*. URL https://code.visualstudio.com/docs. – abgerufen am 2022-08-24

[DuSu08] DUGGAL, GAURAV; SURI, BHARTI: Understanding regression testing techniques. In: *Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology*, 2008

[EbSt11] EBANHESATEN, KHALID; STENHORST, CHRISTIAN: Testautomation in großen Softwareprojekten. In: *HMD Praxis der Wirtschaftsinformatik* Bd. 48 (2011), Nr. 2, S. 88–92. – Company: SpringerDistributor: SpringerInstitution: SpringerLabel: Springernumber: 2publisher: Gabler Verlag

[Eise19] EISELE, OLAF: ROBOTIC PROCESS AUTOMATION (RPA) Mensch-Roboter-Kollaboration in indirekten Bereichen. In: *ifaa – Institut für angewandte Arbeitswissenschaft e. V.* (2019), S. 1–6

[Gith00] *GitHub: Let's build from here*. URL https://github.com/. – abgerufen am 2023-02-25. – GitHub

[GrMT21] GRAF, PATRIK; MEIER, MARKUS A.; TOKARSKI, KIM OLIVER: Anwendung von Robotic Process Automation. In: *Digital Business*, Springer Gabler, Wiesbaden (2021), S. 85–118

[Hard00] HARDER, GABRIEL: Effizienzsteigerung durch automatisierte Ausführung von Softwaretests, S. 83

[Heis21] HEISKANEN, ANDREAS: Robotic Process Automation in automated GUI testing of Web Applications.

[Hers00] HERSCHEL, MARC: Eine Einführung in die Regressionstests, S. 20

[HMPR04] HEVNER, ALAN R.; MARCH, SALVATORE T.; PARK, JINSOO; RAM, SUDHA: Design science in information systems research. In: *MIS quarterly*, JSTOR (2004), S. 75–105

[Holm00] HOLMAN, G. KEN: What is XSLT. In: *XML.com*, 2000

[HoRK08] HORNEGGER, JOACHIM; REIß, JOACHIM; KUWERT, TORSTEN: Softwareentwicklung in der Medizintechnik am Beispiel der medizinischen Bildverarbeitung. In: *Informatik – Forschung und Entwicklung* Bd. 22 (2008), Nr. 3, S. 161–171

[HoSU20] HOFMANN, PETER; SAMP, CAROLINE; URBACH, NILS: Robotic Process Automation. In: *Electronic Markets* Bd. 30 (2020)

[HuKo07] HUIZINGA, DOROTA; KOLAWA, ADAM: *Automated Defect Prevention*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2007 – ISBN 978-0-470-16517-1

[Jack09] JACKSON, RECARDO: Testmanagement: Professionelles Testen. In: *Informatik-Spektrum* Bd. 32 (2009), Nr. 1, S. 37–41. – Company: SpringerDistributor: SpringerInstitution: SpringerLabel: Springernumber: 1publisher: Springer-Verlag

[Jäge08] Frameworks. In: JÄGER, K. (Hrsg.): *Ajax in der Praxis: Grundlagen, Konzepte, Lösungen*, *Xpert.press*. Berlin, Heidelberg: Springer, 2008 – ISBN 978-3-540-69334-5, S. 313–322

[Jmar00] J-MARTENS: *Dokumentation zur Visual Studio-IDE*. URL https://docs.microsoft.com/de-de/visualstudio/ide/. – abgerufen am 2022-08-21

[KoBe10] KOROTKIY, DMITRY; BENDER, KLAUS: Universelle Architektur zur Testautomatisierung. In: *atp magazin* Bd. 52 (2010), Nr. 05, S. 26–31

[KoFe20] KOCH, CHRISTINA; FEDTKE, STEPHEN: *Robotic Process Automation: Ein Leitfaden für Führungskräfte zur erfolgreichen Einführung und Betrieb von Software-Robots im Unternehmen*: Springer-Verlag, 2020. – Google-Books-ID: xjLnDwAAQBAJ – ISBN 978-3-662-61178-4

[LaTu20] LANGMANN, CHRISTIAN; TURI, DANIEL: *Robotic Process Automation (RPA) – Digitalisierung und Automatisierung von Prozessen: Voraussetzungen, Funktionsweise und Implementierung am Beispiel des Controllings und Rechnungswesens*: Springer-Verlag, 2020. – Google-Books-ID: QrPRDwAAQBAJ – ISBN 978-3-658-28299-8

[Lenz06] LENZ, EVAN: *XSLT 1.0 kurz & gut*. 1. Aufl. Köln: O'Reilly Verlag, 2006 – ISBN 3-89721-517-9

[Msxm16] *MSXML Roadmap*. URL https://learn.microsoft.com/en-us/previous-versions/windows/desktop/jj152146(v=vs.85). – abgerufen am 2022-12-16

[Oste16] OSTERHAGE, WOLFGANG: Qualität in der Softwareentwicklung. In: *HMD Praxis der Wirtschaftsinformatik* Bd. 53 (2016), Nr. 2, S. 153–168

[PTRC07] PEFFERS, KEN; TUUNANEN, TUURE; ROTHENBERGER, MARCUS A.; CHATTERJEE, SAMIR: A design science research methodology for information systems research. In: *Journal of management information systems* Bd. 24, Taylor & Francis (2007), Nr. 3, S. 45–77

[Repo00] *Reports und Testdokumentation – QF-Test*. URL https://www.qfs.de/qf-test-handbuch/lc/manual-de-user_report.html#usec_report. – abgerufen am 2022-07-22

[RoSt20] ROBRA-BISSANTZ, SUSANNE; STRAHRINGER, SUSANNE: Wirtschaftsinformatik-Forschung für die Praxis. In: *HMD Praxis der Wirtschaftsinformatik* Bd. 57 (2020), Nr. 2, S. 162–188. – Company: SpringerDistributor: SpringerInstitution: SpringerLabel: Springernumber: 2publisher: Springer Fachmedien Wiesbaden

[Rpaf22] *RPA Framework – RPA Framework documentation*. URL https://rpaframework.org/. – abgerufen am 2022-04-14

[Sieb00] SIEBERT, WALDEMAR: *E2E-Testing: Was sind End-to-End-Tests? Testautomatisierung.org*. URL https://www.testautomatisierung.org/lexikon/e2e-testing/. – abgerufen am 2022-07-27. – Testautomatisierung.org

[SnBS09] SNEED, HARRY M.; BAUMGARTNER, MANFRED; SEIDL, RICHARD: Der Systemtest. In: *Hanser, München* Bd. 2 (2009)

[SnJu11] SNEED, HARRY M.; JUNGMAYR, STEFAN: Mehr Testwirtschaftlichkeit durch Value-Driven-Testing. In: *Informatik-Spektrum* Bd. 34 (2011), Nr. 2, S. 192–209

[SpLi19] SPILLNER, ANDREAS; LINZ, TILO: *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB®-Standard*: dpunkt.verlag, 2019. – Google-Books-ID: le72DwAAQBAJ – ISBN 978-3-96088-501-6

[Task22] TASKT: *System specifications*. URL https://github.com/saucepleez/taskt. – abgerufen am 2022-08-07

[Terr00] TERRYGLEE: *Was ist WPF? – Visual Studio (Windows)*. URL https://docs.microsoft.com/de-de/visualstudio/designers/getting-started-with-wpf. – abgerufen am 2022-08-21

[Uipa00a] *UiPath Studio*. URL https://docs.uipath.com/studio/lang-de/docs/about-automation-projects. – abgerufen am 2022-07-23. – UiPath Studio

[Uipa00b] UIPATH: *Automatisierungsplattform – Führendes RPA-Unternehmen | UiPath*. URL https://www.uipath.com/de. – abgerufen am 2022-08-07

[Uipa00c] UIPATH: *Erstellen eines Basisprozesses*. URL https://docs.uipath.com/studio/lang-de/docs/creating-basic-process. – abgerufen am 2022-08-07. – UiPath Studio

[Vija07] VIJAY: *What is Regression Testing? Definition, Tools, Method, and Example*. URL https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/. – abgerufen am 2022-07-26. – Software Testing Help

[VPTB00] VERAS, PAULO; PESTITSCHEK, THIAGO; THIJSSEN, DIEGO; BENTIVOGLIO, TIAGO: *BPA/OpenIAP Docs – Integrated Framework Documentation – BPA/OpenIAP Docs 1.0.5 documentation*. URL https://docs.openiap.io/. – abgerufen am 2022-08-07

[WaAL07] WAGNER, JULIA; ANDRES, THOMAS; LAUER, YVES: Vom Prozess zur Ausführung – Modellgestützte Entwicklung betriebswirtschaftlicher Software. In: BECKER, J.; DELFMANN, P.; RIEKE, T. (Hrsg.): *Effiziente Softwareentwicklung mit Referenzmodellen*. Heidelberg: Physica-Verlag HD, 2007 – ISBN 978-3-7908-1994-6, S. 61–76

[Wasi00] *Was ist ACL Analytics?* URL https://help.highbond.com/helpdocs/analytics/13/user-guide/de/Content/ui/what_is_acl_analytics.htm. – abgerufen am 2022-11-28

[WiHe06] WILDE, THOMAS; HESS, THOMAS: *Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung*: Arbeitsbericht, 2006

[Witt19] WITTE, FRANK: *Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019 – ISBN 978-3-658-25086-7

[Zwac17] ZWACK, THOMAS: Gestaltungsorientierter Forschungsansatz. In: *Peer-to-Peer-Geschäftsmodelle zur Absicherung privater Risiken: Eine Exploration am Beispiel Wildschaden*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017 – ISBN 978-3-658-18315-8, S. 13–19

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Anf.	Anforderung
CSS	Cascading Style Sheets
DLL	Dynamic Link Library
eEPK	Erweiterte ereignisgesteuerte Prozesskette
HTML	Hypertext Markup Language
MSXML	Microsoft XML Core Services
RPA	Robotic Process Automation
UML	Unified Modeling Language
WPF	Windows Presentation Foundation
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Anhang A

```
<?xml version="1.0" encoding="utf-8"?>
<TestcaseXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Bezeichnung>Import_BHDaten</Bezeichnung>
<Bereich>Datenimport</Bereich>
<SkriptID>0b6e3a59-f898-45bc-9b82-14d181fb9ef4</SkriptID>
<SkriptPfad />
 <AblagePfad>D:\Daten\Framework_RPA\Testfälle\Datenimport\Import_BHDaten
</AblagePfad>
<DatenPfad>
D:\Daten\Framework_RPA\Testfälle\Datenimport\Import_BHDaten\Import_BHDaten.xlsx
</DatenPfad>
<ErgebnisXML>
D:\Daten\Framework_RPA\Ergebnisse\01012099_000000\Import_BHDaten.xml
</ErgebnisXML>
</TestcaseXML>
```

Anhang B

```xml
<?xml version="1.0" encoding="utf-8"?>
<SingleResult xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Bezeichnung>Mustertestfall</Bezeichnung>
  <Bereich>Muster</Bereich>
  <Fortschritt>
    <Step TimeStamp="01.01.2099 15:15:31">Beschreibung 1</Step>
    <Step TimeStamp="01.01.2099 15:15:33">Beschreibung 2</Step>
    <Step TimeStamp="01.01.2099 15:15:34">Beschreibung 3</Step>
  </Fortschritt>
  <Ergebnisse>
    <Ergebnis
        TimeStamp="01.01.2099 15:15:35"
        Soll="1"
        Ist=" 1"
        Result="OK">
        Ergebnisbeschreibung 1
    </Ergebnis>
  </Ergebnisse>
  <Start>01.01.2099 15:15:13</Start>
  <Ende>01.01.2099 15:15:35</Ende>
  <Dauer>00:00:22</Dauer>
  <GesamtErgebnis>OK</GesamtErgebnis>
  <HistorischesErgebnis Zeitpunkt="01012099_000000" Ergebnis="OK" />
</SingleResult>
```

Anhang C

C# Code der Klasse RPATestautomation aus der Testautomation.dll, welche die Methoden für die Erweiterung der RPA-Software für die Testautomatisierung enthält.

```
1   public class RPATestautomation
2       {
3           public bool WriteProgress(
4                       string pathXMLFile,string text)
5           {
6               //XML laden
7               XmlDocument doc = new XmlDocument();
8               doc.Load(pathXMLFile);
9               XmlNode root = doc.DocumentElement;
10              //Element Fortschritt laden
11              XmlNode entries = root.SelectSingleNode("Fortschritt");
12
13              //Neuen Eintrag mit Datum und Uhrzeit erstellen
14              XmlElement elem = doc.CreateElement("Step");
15              elem.SetAttribute("TimeStamp", DateTime.Now.ToString());
16              elem.InnerText = text;
17
18              //Neuen Eintrag dem Element Fortschritt hinzufügen
19              entries.AppendChild(elem);
20              doc.Save(pathXMLFile);
21              return true;
22          }
23          public bool WriteResult(
24                      string pathXMLFile, string text, string soll, string ist)
25          {
26              //XML laden
27              XmlDocument doc = new XmlDocument();
28              doc.Load(pathXMLFile);
29              XmlNode root = doc.DocumentElement;
30              //Element Ergebnisse laden
31              XmlNode entries = root.SelectSingleNode("Ergebnisse");
32
33              //Neuen Eintrag mit Datum und Uhrzeit erstellen
34              XmlElement elem = doc.CreateElement("Ergebnis");
35              elem.SetAttribute("TimeStamp", DateTime.Now.ToString());
36              //Soll, Ist und die Beschreibung hinzufügen
37              elem.SetAttribute("Soll", soll);
38              elem.SetAttribute("Ist", ist);
39              elem.InnerText = text;
40
41              //Neuen Eintrag dem Element Ergebnisse hinzufügen
42              entries.AppendChild(elem);
43              doc.Save(pathXMLFile);
44              return true;
```

```
45          }
46          public string GetValueFromTestcaseXML(
47                          string pathXMLFile,string value)
48          {
49              //XML laden
50              XmlDocument doc = new XmlDocument();
51              doc.Load(pathXMLFile);
52              XmlNode root = doc.DocumentElement;
53              //Den gesuchten Eintrag aus der TestcaseXML zurückgeben
54              return root.SelectSingleNode(value).InnerText;
55          }
56          public bool SetError(string pathXMLFile)
57          {
58              //XML laden
59              XmlDocument doc = new XmlDocument();
60              doc.Load(pathXMLFile);
61              XmlNode root = doc.DocumentElement;
62
63              //Neuen Eintrag GesamtErgebnis erstellen
64              XmlElement elem = doc.CreateElement("GesamtErgebnis");
65              //Ergebnis des Testdurchlaufes auf Error setzen
66              elem.InnerText = "Error";
67
68              //Eintrag hinzufügen
69              root.AppendChild(elem);
70              doc.Save(pathXMLFile);
71              return true;
72          }
73          public void CloseRPA()
74          {
75              //Softwareroboter beenden
76              Environment.Exit(0);
77          }
78      }
```

Anhang D

XSLT-Stylesheet für die Transformation der XML der einzelnen Ergebnisse zu der HTML mit den Detailergebnissen für den Testfall im Test-Reporting.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-
prefixes="msxsl">
    <xsl:output encoding="utf-8" method="html" version="4.0" indent="no"/>

    <xsl:template match="/">
        <xsl:comment> saved from url=(0016)http://localhost </xsl:comment>
        <xsl:text>&#xD;&#xA;</xsl:text>
        <html lang="de">
            <head>
                <title>Testreport</title>
            </head>
            <body>
                <div>
                    <div>
                    <!-- Zurück Button für die Navigation -->
                        <input type="button"
                                value="zurück"
                                onclick="history.back();"/>
                    </div>
                    <div style="margin: 20px;
                                font-size: 24px;
                                font-weight: bold;
                                width: 100%;">
                    <!-- Hier wird die Überschrift für das
                         Test-Reporting erstellt -->
                    <div style="text-align: center;">
                        <xsl:value-of
                            select="SingleResult/Bereich"/>
                    </div>
                    <div style="text-align: center;">
                        <xsl:value-of
                            select="SingleResult/Bezeichnung"/>
                    </div>
                    </div>
                </div>
            <!-- CSS-Regeln für die Formatierung -->
            <style>
                    body {
                        font-size: 18px;
                    }

                    table {
                        border-collapse: collapse;
```

```
                    }

                    th {
                        background-color: #6aff80;
                        font-weight: bold;
                        color: #000000;
                        border: 1px solid black;
                    }

                    td {
                        text-align: center;
                        border: 1px solid black;
                    }

                    .td_OK {
                        background-color: #8AFA76;
                        text-align: center;
                    }

                    .td_FAIL {
                        background-color: #FF5258;
                        text-align: center;
                    }

                    .td_Error {
                        background-color: #808080;
                        text-align: center;
                    }

                    textarea {
                        resize: none;
                        border: none;
                        outline: none;
                    }
            </style>
                <xsl:apply-templates select="SingleResult"/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="SingleResult">
        <div style="display: flex;">
            <div style="width: 100%;">
                <div style="display: flex;
                            align-items: center;">
                    <!-- Der Bereich mit der Dauer wird eingefügt -->
                    <xsl:call-template name="Dauer"/>
                </div>
                <div style="display: flex; margin-bottom: 50px;">
                    <!-- Der linke Bereich mit den Ergebnissen wird erstellt -
->
```

```
                    <xsl:apply-templates select="Ergebnisse"/>
                    <!-- Der rechte Bereich mit den Fortschritten wird
                        erstellt -->
                    <xsl:apply-templates select="Fortschritt"/>
                </div>
            </div>
        </div>
    </xsl:template>
    <xsl:template match="Fortschritt">
        <div style="margin-left: 10px; width: 40%;">
            <table style="margin-left: 10px; width: 100%;">
                <caption style="font-weight: bold;
                                font-size: x-large;">
                               Protokoll</caption>
                <!-- Spaltenüberschriften hinzufügen -->
                <tr>
                    <th>TimeStamp</th>
                    <th>Hinweis</th>
                </tr>
                <!-- Die einzelnen Einträge erstellen -->
                <xsl:for-each select="Step">
                    <tr>
                        <td>
                            <xsl:value-of select="@TimeStamp"/>
                        </td>
                        <td>
                            <xsl:value-of select="."/>
                        </td>
                    </tr>
                </xsl:for-each>
            </table>
        </div>
    </xsl:template>
    <xsl:template match="Ergebnisse">
        <div style="margin-left: 50px; width: 40%;">
            <table style="margin-left: 10px; width: 100%;">
            <caption style="font-weight: bold;
                            font-size: x-large;">
                            Ergebnisse</caption>
                <!-- Spaltenüberschriften hinzufügen -->
                <tr>
                    <th>TimeStamp</th>
                    <th>Hinweis</th>
                    <th>Soll</th>
                    <th>Ist</th>
                    <th>Ergebnis</th>
                </tr>
                <!-- Die einzelnen Einträge erstellen -->
                <xsl:for-each select="Ergebnis">
                    <tr>
```

```
                        <td>
                            <xsl:value-of select="@TimeStamp"/>
                        </td>
                        <td>
                            <xsl:value-of select="."/>
                        </td>
                        <td>
                            <xsl:value-of select="@Soll"/>
                        </td>
                        <td>
                            <xsl:value-of select="@Ist"/>
                        </td>
                        <xsl:element name="td">
                            <xsl:choose>
                                <xsl:when test="@Result='OK'">
                                    <xsl:attribute
                                        name="class">
                                    td_OK
                                    </xsl:attribute>
                                </xsl:when>
                                <xsl:when
                                    test="@Result='FAIL'">
                                    <xsl:attribute
                                        name="class">
                                        td_FAIL
                                    </xsl:attribute>
                                </xsl:when>
                                <xsl:when
                                    test="@Result='Error'">
                                    <xsl:attribute
                                        name="class">
                                        td_Error
                                    </xsl:attribute>
                                </xsl:when>
                            </xsl:choose>
                            <xsl:value-of select="@Result"/>
                        </xsl:element>
                    </tr>
                </xsl:for-each>
            </table>
        </div>
    </xsl:template>
    <xsl:template name="Dauer">
        <div>
            <div style="margin: 20px;">
            <!-- Die einzelnen Bereiche mit Start, Ende
                und Dauer werden erstellt -->
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Start
```

```
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Start"/>
                    </div>
                </div>
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Ende
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Ende"/>
                    </div>
                </div>
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Dauer
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Dauer"/>
                    </div>
                </div>
            </div>
        </div>
    </xsl:template>
</xsl:stylesheet>
```

Anhang E

XSLT-Stylesheet für die Transformation der XML mit dem Gesamtergebnis zu der HTML mit der Übersicht im Test-Reporting.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3      xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-
4  prefixes="msxsl">
5      <xsl:output encoding="utf-8" method="html" version="4.0" indent="no"/>
6      <xsl:template match="/">
7          <xsl:comment> saved from url=(0016)http://localhost </xsl:comment>
8          <xsl:text>&#xD;&#xA;</xsl:text>
9          <html lang="de">
10             <head>
11                 <title>Testreport</title>
12             </head>
13             <!-- Beim Laden der Datei wird das Diagramm erstellt -->
14             <body onload="setChart();">
15                 <div style="display: flex;">
16                     <div>
17                     <!-- Zurück Button für die Navigation -->
18                         <input type="button"
19                                value="zurück"
20                                onclick="history.back();"/>
21                     </div>
22                     <div style="margin: 20px;
23                                 font-size: 24px;
24                                 font-weight: bold;
25                                 width: 100%;">
26                     <!-- Hier wird die Überschrift für das
27                          Test-Reporting erstellt -->
28                         <div style="text-align: center;">
29                             Testreport
30                         </div>
31                         <div style="text-align: center;">
32                             <xsl:value-of
33                             select="TotalResult/Start"/>
34                         </div>
35                     </div>
36                 </div>
37                 <script src="https://cdn.jsdelivr.net/npm/chart.js">
38                 </script>
39                 <!-- Die Daten der Ergebnisse werden in die Javascript
40                      Variable daten geschrieben.
41                      Diese Variable wird für die Erstellung des Diagrammes
42                      benötigt. -->
43                 <script type="text/javascript">
44                 <xsl:text>
```

```
                    var daten = [
                    </xsl:text>
                    <xsl:value-of select="TotalResult/GesamtOK"/>
                    <xsl:text>,</xsl:text>
                    <xsl:value-of select="TotalResult/GesamtFail"/>
                    <xsl:text>,</xsl:text>
                    <xsl:value-of select="TotalResult/GesamtError"/>
                    <xsl:text>
                    ];
                    </xsl:text>
                </script>

                <!-- Dieser Bereich enthält Javascript Funktionen mit denen
                        das Diagramm erstellt wird. -->
                <script type="text/javascript"><![CDATA[
            function setChart() {

            //Balkenbeschriftung
                const labels = [
                    'OK',
                    'FAIL',
                    'Error'
                ];

            //Balken: Beschriftung, Farben, maximale Breite und Daten
              hinzufügen
                const data = {
                    labels: labels,
                    datasets: [{
                        label: "",
                        backgroundColor: ["#6aff80", "#FF5258", "#808080"],
                        borderColor: ["#6aff80", "#FF5258", "#808080"],
                        maxBarThickness: 30,
                        data: daten,
                    }]
                };

            //Diagramm: Fertige Balken hinzufügen und erforderliche
              Eigenschaften festlegen
                const config = {
                    type: 'bar',
                    data: data,
                    options: {
                        plugins: {
                            legend: {
                                display: false
                            }
                        },
                        scales: {
                            y: {
```

```
                            ticks: {
                                beginAtZero: true,
                                stepSize: 1
                            }
                        }
                    }
                }
            };

        //Diagramm erstellen
            const myChart = new Chart(
                document.getElementById('myChart'),
                config
            );
        }
    ]]></script>
    <!-- CSS-Regeln für die Formatierung -->
            <style>
                body {
                     font-size: 18px;
                }

                table {
                      border-collapse: collapse;
                }

                th {
                      background-color: #6aff80;
                      font-weight: bold;
                      color: #000000;
                      border: 1px solid black;
                }

                td {
                      text-align: center;
                      border: 1px solid black;
                }

                .td_OK {
                      background-color: #8AFA76;
                      text-align: center;
                }

                .td_FAIL {
                      background-color: #FF5258;
                      text-align: center;
                }

                .td_Error {
                      background-color: #808080;
```

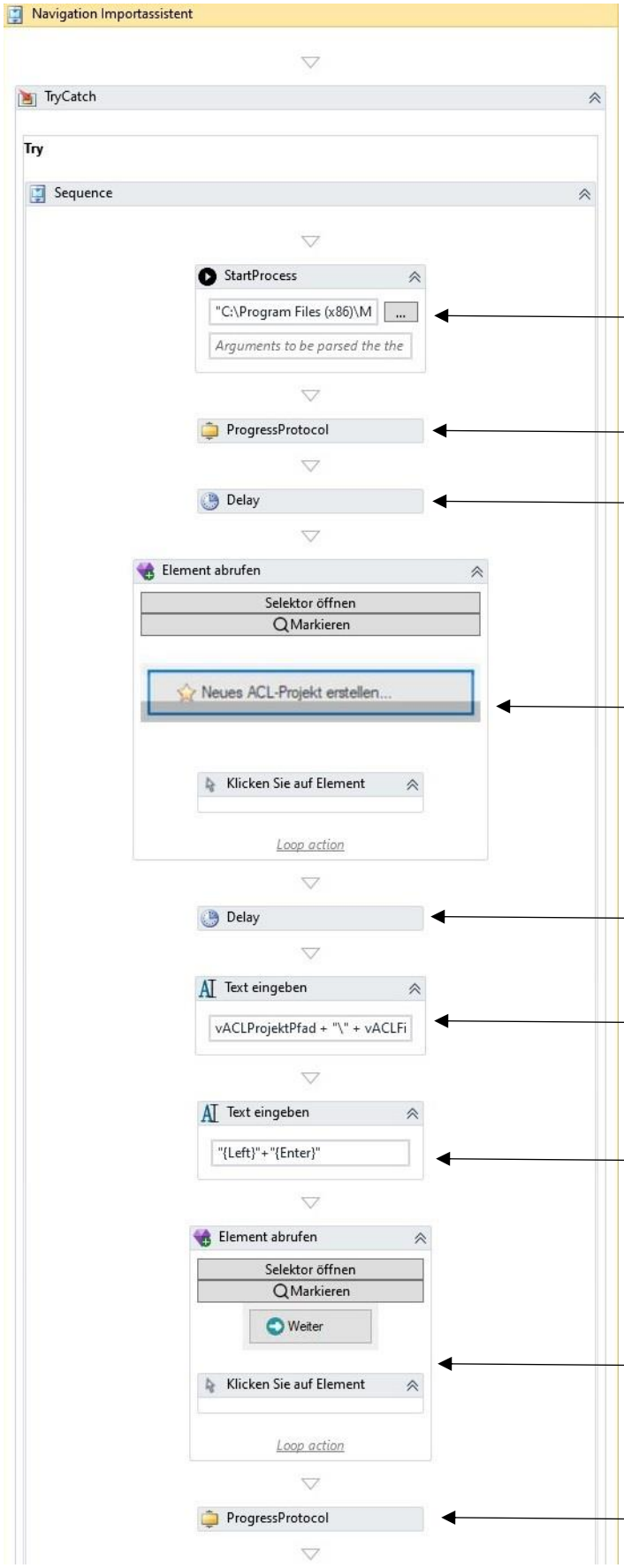
```
                        text-align: center;
                    }

                    textarea {
                        resize: none;
                        border: none;
                         outline: none;
                    }
                </style>
                <xsl:apply-templates select="TotalResult"/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="TotalResult">
        <div style="display: flex;">
            <div>
                <div style="display: flex;
                            align-items: center;
                            margin-bottom: 50px;">
                    <!-- Der Bereich mit der Dauer wird eingefügt -->
                    <xsl:call-template name="Dauer"/>
                    <!-- Der Bereich mit dem Diagramm wird eingefügt -->
                    <xsl:call-template name="Ergebnisübersicht"/>
                </div>
                <!-- Der Bereich mit den Ergebnissen wird eingefügt -->
                <xsl:apply-templates select="Ergebnisse"/>
            </div>
            <div style="margin: 10; margin-left: 50;">
            <!-- Der Bereich mit den Systeminformationen wird eingefügt -->
                <xsl:call-template name="Systeminfo"/>
            </div>
        </div>
    </xsl:template>
    <xsl:template name="Systeminfo">
        <textarea rows="35" cols="50" readonly="true" wrap="false">
            <!-- Dieser Bereich wird nur erstellt wenn eine
                Anmerkung eingetragen wurde -->
            <xsl:if test="not(Anmerkung='')">
            <xsl:text>
            ------------Anmerkung zum Testdurchlauf------------
            </xsl:text>
            <!-- Anmerkung wird eingefügt -->
            <xsl:value-of select="Anmerkung"/>
            <xsl:text>
            ------------------------------------------------------
            </xsl:text>
            </xsl:if>
            <!-- Systeminformationen werden eingefügt -->
            <xsl:value-of select="Systeminfo"/>
        </textarea>
```

```
195         </xsl:template>
196     <xsl:template match="Ergebnisse">
197     <div>
198         <table style="margin-left: 10px; width: 100%;">
199             <!-- Spaltenüberschriften hinzufügen -->
200             <tr>
201                 <th></th>
202                 <th>Testfall</th>
203                 <th>Ergebnis</th>
204                 <!-- Diese Spaltenüberschriften werden nur erstellt wenn
205                     historische Ergebnisse ausgewählt wurden -->
206                 <xsl:if test="SingleResultsforTotalResult/HistErgebnis">
207                     <th>Historisches Ergebnis<br/>vom
208                         <xsl:value-of
209                   select="SingleResultsforTotalResult/HistZeitpunkt"/>
210                     </th>
211                     <th>
212                         <xsl:element name="a">
213                             <xsl:attribute name="href">
214                                 <xsl:text>../</xsl:text>
215                                 <xsl:value-of
216                 select="SingleResultsforTotalResult/HistZeitpunkt"/>
217                                 <xsl:text>/Ergebnis_</xsl:text>
218                                 <xsl:value-of
219                 select="SingleResultsforTotalResult/HistZeitpunkt"/>
220                                 <xsl:text>.html</xsl:text>
221                             </xsl:attribute>
222                             Hist. Testreport
223                         </xsl:element>
224                     </th>
225                 </xsl:if>
226             </tr>
227             <!-- Die einzelnen Einträge erstellen -->
228             <xsl:for-each select="SingleResultsforTotalResult">
229             <tr>
230                 <td>
231                     <xsl:element name="a">
232                         <xsl:attribute name="href">
233                             <xsl:value-of select="Bezeichnung"/>
234                             <xsl:text>.html</xsl:text>
235                         </xsl:attribute>
236                         Details
237                     </xsl:element>
238                 </td>
239                 <td><xsl:value-of select="Bezeichnung"/></td>
240                 <xsl:element name="td">
241                 <!-- Je nach Ergebnis wird die entsprechende
242                         Formatierung zugeordnet -->
243                     <xsl:choose>
244                         <xsl:when test="Ergebnis='OK'">
```

```
                        <xsl:attribute name="class">
                        td_OK
                    </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="Ergebnis='FAIL'">
                        <xsl:attribute name="class">
                        td_FAIL
                    </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="Ergebnis='Error'">
                        <xsl:attribute name="class">
                        td_Error
                    </xsl:attribute>
                    </xsl:when>
                </xsl:choose>
                <xsl:value-of select="Ergebnis"/>
            </xsl:element>
            <!-- Dieser Bereich wird nur erstellt wenn
                historische Ergebnisse ausgewählt wurden -->
            <xsl:if test="HistErgebnis">
                <xsl:element name="td">
                <xsl:choose>
                    <xsl:when test="HistErgebnis='OK'">
                        <xsl:attribute name="class">
                        td_OK
                    </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="HistErgebnis='FAIL'">
                        <xsl:attribute name="class">
                        td_FAIL
                    </xsl:attribute>
                    </xsl:when>
                    <xsl:when test="HistErgebnis='Error'">
                        <xsl:attribute name="class">
                        td_Error
                    </xsl:attribute>
                    </xsl:when>
                </xsl:choose>
                <xsl:value-of select="HistErgebnis"/>
            </xsl:element>
                <td>
                <!-- Den Link zum historischen Ergebnis erstellen -->
                    <xsl:element name="a">
                        <xsl:attribute name="href">
                            <xsl:text>../</xsl:text>
                            <xsl:value-of
                            select="HistZeitpunkt"/>
                            <xsl:text>/</xsl:text>
                            <xsl:value-of
                            select="Bezeichnung"/>
```

```
                                <xsl:text>.html</xsl:text>
                            </xsl:attribute>
                            Hist. Details
                        </xsl:element>
                    </td>
                </xsl:if>
            </tr>
            </xsl:for-each>
        </table>
    </div>
    </xsl:template>
    <xsl:template name="Dauer">
        <div>
            <div style="margin: 20px;">
            <!-- Die einzelnen Bereiche mit Start, Ende
                und Dauer werden erstellt -->
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Start
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Start"/>
                    </div>
                </div>
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Ende
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Ende"/>
                    </div>
                </div>
                <div style="display: flex;">
                    <div style="width: 40px;">
                        Dauer
                    </div>
                    <div style="margin-left: 10px;">
                        <xsl:value-of select="Dauer"/>
                    </div>
                </div>
            </div>
        </div>
    </xsl:template>
    <xsl:template name="Ergebnisübersicht">
        <div style="width: 400px; margin-left: 30px;">
            <!-- Hier wird das Element für das Diagramm erstellt -->
            <canvas id="myChart"></canvas>
        </div>
    </xsl:template>
</xsl:stylesheet>
```

Anhang F



Testobjekt starten

Fortschritt für das Framework schreiben

2 Sekunden warten, bis die Benutzeroberfläche geladen ist

Schaltfläche erkennen und betätigen

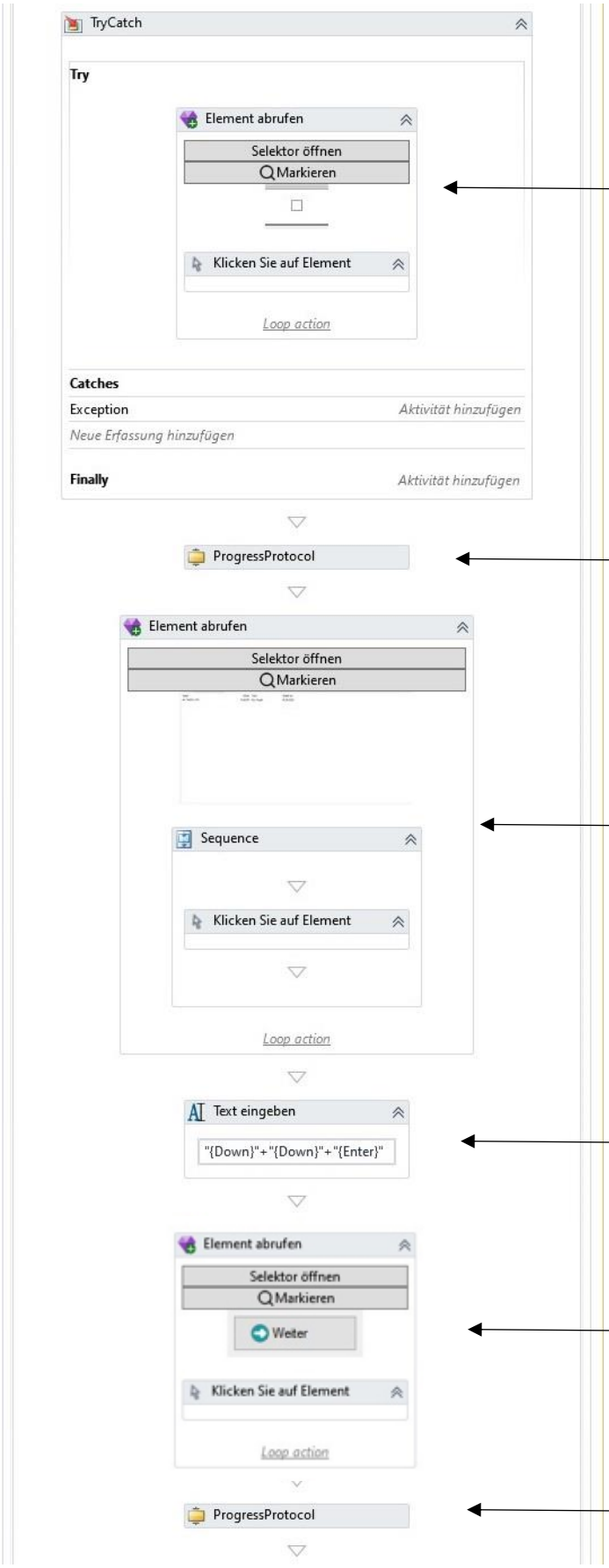
2 Sekunden warten, bis das Dialogfeld geöffnet ist

Eingabe von einem Pfad anhand von Variablen und Zeichen

Mit den Tasten Pfeil-Links und Enter das Dialogfeld schließen

Schaltfläche erkennen und betätigen

Fortschritt für das Framework schreiben



Checkbox erkennen und betätigen

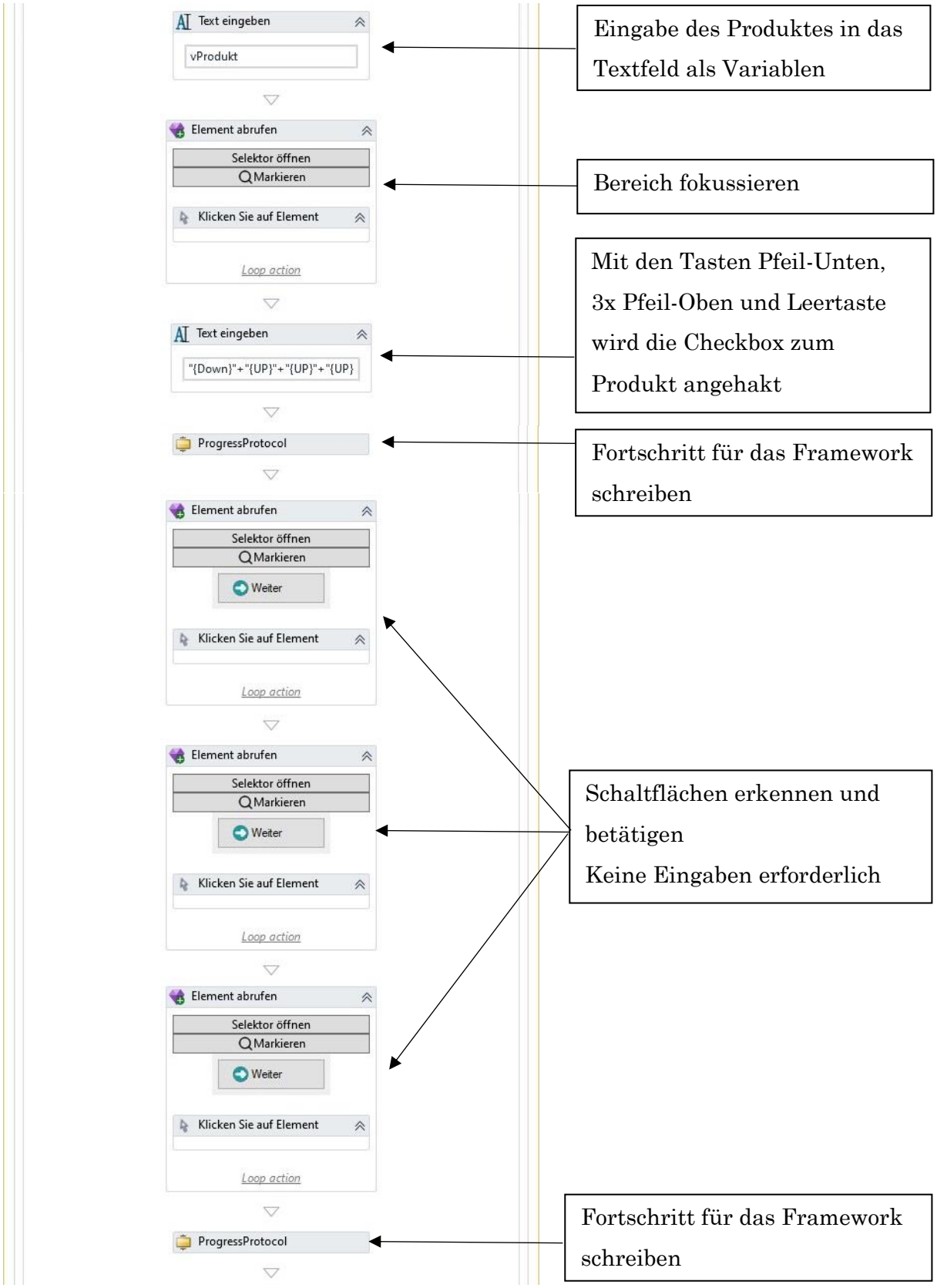
Fortschritt für das Framework schreiben

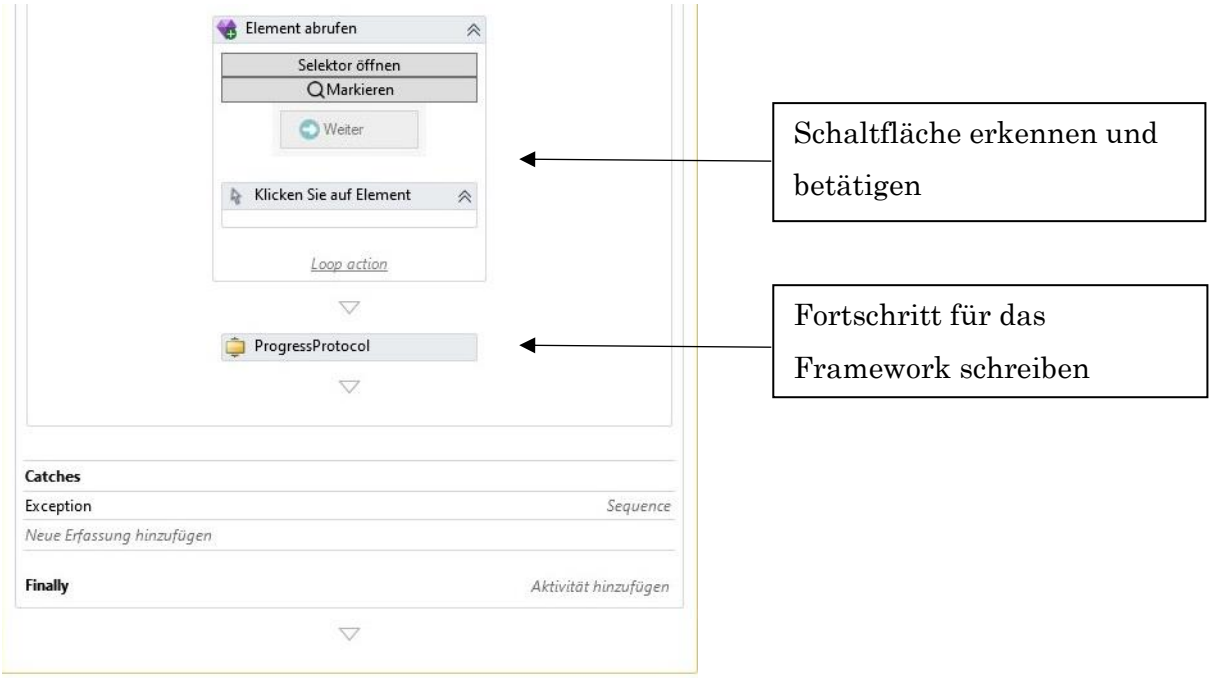
Bereich erkennen und rechte Maustaste betätigen

Mit den Tasten 2x Pfeil-Unten und Enter im Kontextmenü „Alle Auswählen“ auswählen

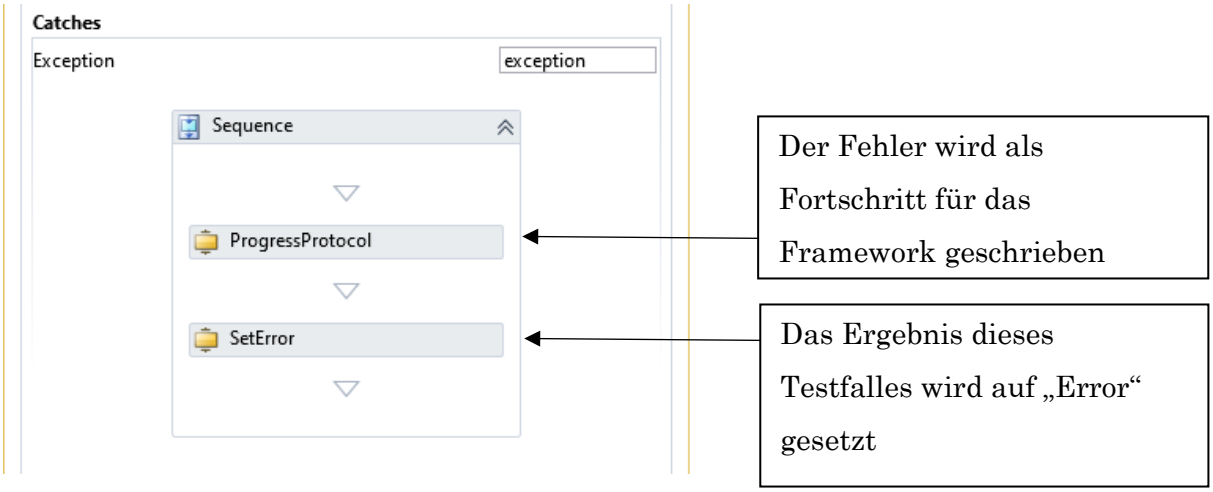
Schaltfläche erkennen und betätigen

Fortschritt für das Framework schreiben


Text eingeben
vProdukt
Element abrufen
Selektor öffnen
Markieren
Klicken Sie auf Element
Loop action
Text eingeben
"{Down}"+"{UP}"+"{UP}"+"{UP}
ProgressProtocol
Element abrufen
Selektor öffnen
Markieren
Weiter
Klicken Sie auf Element
Loop action
Element abrufen
Selektor öffnen
Markieren
Weiter
Klicken Sie auf Element
Loop action
Element abrufen
Selektor öffnen
Markieren
Weiter
Klicken Sie auf Element
Loop action
ProgressProtocol
Eingabe des Produktes in das Textfeld als Variablen
Bereich fokussieren
Mit den Tasten Pfeil-Unten, 3x Pfeil-Oben und Leertaste wird die Checkbox zum Produkt angehakt
Fortschritt für das Framework schreiben
Schaltflächen erkennen und betätigen
Keine Eingaben erforderlich
Fortschritt für das Framework schreiben



Sollte es bei dem oben angeführten Prozess zu einem Fehler kommen, wird die Exception abgefangen und dieser Prozess ausgeführt.



Anhang G

```
Try
    // Comment: Testobjekt starten
    Start Process [Process: C:\Program Files (x86)\Mentor\EinlesenFV\Einlesen_FV.exe]
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    Pause Script [Wait for 5000ms]
    Set Window State [Target Window: Current Window, Window State: Maximize]
    Pause Script [Wait for 1000ms]
    UI Automation [Left Click element in window 'Daten einlesen - ACL-Projekt wählen']
    Send Keystrokes [Send '{vACLProjektPfad} \{vACLFile}{Enter}' to 'Current Window']
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    // Comment: Wählt die Daten für den Import aus
    UI Automation [Right Click element in window 'Daten einlesen - Dateien wählen']
    Send Advanced Keystrokes [Send To Window 'Current Window']
    Pause Script [Wait for 500ms]
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    // Comment: Filtert nach Produkt und wählt aus
    Pause Script [Wait for 1000ms]
    Send Keystrokes [Send '{vProdukt}' to 'Current Window']
    Send Mouse Move [Target Coordinates (229,141) Click: Left Click]
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    Pause Script [Wait for 3000ms]
    // Comment: Jahr wählen
        If ({vJahrRelevant} is equal to 1)
            Send Mouse Move [Target Coordinates (674,{vAbstand}) Click: Left Click]
                Loop {vAnzahlDateien} Times
                    Send Keystrokes [Send '{ENTER}' to 'Current Window']
                    Set Variable [Apply '{vAbstand} + 26' to Variable 'vAbstand']
                End Loop
        End If
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    Pause Script [Wait for 500ms]
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
    Pause Script [Wait for 5000ms]
    // Comment: Schaltfläche Weiter
    Send Keystrokes [Send '%w' to 'Current Window']
    // Comment: Prüft ob der Importprozess gestartet wurde.
    Wait For Window To Exist [Target Window: '{vACLFile}.ACL - Analytics', Wait Up To 120 seconds]
        Loop While GUI Element Exists [Find TitleBar Element In TestACL.ACL - Analytics]
            Pause Script [Wait for 2000ms]
        End Loop
    // Comment: Fortschritt für das Framework schreiben
    Execute DLL [Call Method 'Boolean WriteProgress(System.String, System.String)' in 'Testautomation.RPATestautomation']
Catch Exception [Items in this section will run if error occurs]
    Set Variable [Apply '1' to Variable 'vResult']
    Stop Current Task
End Try
```