Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Torben Kuhlmann

Automatisierung von Standardtestfällen der Anwendungslogik eines ERP-Systems

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Torben Kuhlmann

Automatisierung von Standardtestfällen der Anwendungslogik eines ERP-Systems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 7. Juni 2018

Torben Kuhlmann

Thema der Arbeit
Automatisierung von Standardtestfällen der Anwendungslogik eines ERP-Systems

Stichworte
Automatiserung der Testausführung, ERP, ERP-System, Systemtest, AutoIt, geschäftsprozessbasierter Test, Werkzeugeinführung

Kurzzusammenfassung
Im Rahmen dieser Arbeit wird die Einführung eines Werkzeuges für ein ERP-System beschrieben und analysiert. Das Werkzeug ist für den internen Einsatz zur Unterstützung der manuellen Testaktivitäten gedacht. Es sollen damit Standardtestfälle zur Prüfung der Kernfunktionalität in Form von geschäftsprozessbasierten Tests umgesetzt werden. Die aus diesen Testfällen resultierenden Testskripte sollen dabei die grafische Oberfläche des ERP-Systems steuern. Mithilfe des Werkzeuges wird eine Verbesserung in Form einer Zeitersparnis und Erhöhung der Qualität anvisiert.

Torben Kuhlmann

Title of the paper
Automating standard test cases on business logic level for an ERP system

Keywords
Automation of test execution, ERP, ERP System, System testing, AutoIt, business process-based testing, tool introduction

Abstract
Within this thesis the introduction of a tool for an ERP system is described and analyzed. The tool is intended for internal use to support manual testing activities. It will be used to implement standard test cases for testing core functionality in the form of business process-based testing. The test scripts resulting from these test cases should control the graphical user interface of the ERP system. With the help of the tool an improvement of time saving and an increase of quality is targeted.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

[SBS12] nennen Automatisierung als aktuellen Ansatz zur Effizienzsteigerung von Softwaretests. Sie identifizieren als Ziele für eine Effizienzsteigerung nicht nur die Ausführung von mehr Tests, sondern auch die Ermittlung, die Aufbereitung und die Auswertung der Testfälle zu automatisieren, anstatt mehr Personal für das manuelle Testen einzustellen.

Gründe, warum die Durchführung von Tests automatisiert werden, sind:

1. Die Verbesserung der Effizienz von Tests [LS12].
2. Die Verbesserung der Zuverlässigkeit von Tests [LS12].
3. Die Schaffung neuer Fähigkeiten [LS12].

Mit der Hilfe von Testwerkzeugen können manuelle Tätigkeiten, wie sich wiederholende und zeitaufwendige Tätigkeiten automatisiert werden, um eine Verbesserung der Effizienz der Tests zu erreichen [LS12]. Die Durchführung durch das Werkzeug kann dann für einen erhöhten Durchsatz von Testfällen sorgen, was zu einer Ersparnis von Personalressourcen führt [WSLR14]. Werden Ressourcen frei, kann sich das Testpersonal stattdessen anderen Testaufgaben widmen, die mehr Kreativität erfordern.

Die Zuverlässigkeit bei der Durchführung einfacher, sich wiederholender Tätigkeiten durch ein Werkzeug wird erhöht [LS12]. Gegenüber Testwerkzeugen, also Maschinen, können Menschen ermüden oder sie werden während der Arbeit abgelenkt. Beides sind Situationen, in denen häufig Fehler passieren.

Es lassen sich neue Fähigkeiten schaffen, die ohne Automatisierung vorher nicht möglich oder nur mit viel Aufwand umsetzbar waren. Als Beispiele seien hier Tests zur Prüfung des Systems unter Last oder Performanztests genannt [LS12].

1.1 Problemstellung

Die Firma PASCAL Beratungsgesellschaft für Datenverarbeitung m.b.H. hat sich zur Prüfung des ERP-Systems (Enterprise-Resource-Planning-System) INTEGRA® dafür entschieden, im Bereich des Systemtests und der Testautomatisierung ein Werkzeug einzuführen, um den bisherigen manuellen Testprozess zu unterstützen.

Das bisherige Vorgehen beim manuellen Testen im Bereich des Systemtests umfasst:

- Manuelles Testen der Anwendung durch den Entwickler aus Sicht des Endanwenders nach Vorgaben.
- Manuelles Testen der Anwendung durch den Projektleiter aus Sicht des Endanwenders nach Vorgaben.

Im Rahmen dieser Arbeit soll ein Werkzeug entwickelt werden, das die Durchführung manueller Endanwendertests automatisiert. Das heißt, es geht um die Automatisierung der Testdurchführung durch ein Werkzeug.

Werkzeuge, die hier Anwendung finden können, sind solche zur Automatisierung der Interaktion mit der Oberfläche des Testobjekts, sogenannte:

- Capture & Replay Tools: Diese Werkzeuge arbeiten nach dem Prinzip eines Videorekorders [LS12]. Das Werkzeug wird dazu in den Aufnahmemodus versetzt. Während der Aufnahme werden die gewünschten Aktivitäten auf dem Testobjekt ausgeführt. Das Ergebnis ist ein Testskript, dass mit dem Werkzeug zur Wiederausführung gebracht wird.
- Skriptbasiertes Testen: Darunter versteht man, die Durchführung einer vorher festgelegten und dokumentierten Abfolge von Testschritten in einer Skriptsprache [GTBe17]. Das hier erzeugte Testskript wird mit den Sprachmitteln oder einem Testausführungswerkzeug zur Ausführung gebracht.

Die Firma PASCAL möchte das Werkzeug später für mehrere Zwecke einsetzen:

- Mit dem Werkzeug sollen Testfälle in Form von komplexen Geschäftsprozessen erstellt und ausgeführt werden können. Die Testfälle sollen dabei die Kernfunktionalität der Geschäftsprozesse von Kunden abbilden. Das Werkzeug soll dabei die Arbeitsweise der

oben beschriebenen Werkzeuge (Capture & Replay oder skriptbasiertes Testen) haben. Die Schnittstelle zur Interaktion ist also die grafische Oberfläche des ERP-Systems. Die Erstellung von Testskripten wird in der Test-Umgebung der Firma PASCAL durchgeführt.

- Die Ausführung der erzeugten Testskripte sollen bei Bedarf manuell angestoßen werden können. Der Ausführungsort ist dann die Test-Umgebung.
- Die Testskripte sollen als Regressionstests in der Build-Umgebung von PASCAL integriert werden. Die Testskripte sollen nach den nächtlichen Build-Vorgängen laufen. Dazu sollen sie in die Build-Skripte integriert werden.
- Mit dem Werkzeug soll die Durchführung von Lasttests ermöglicht werden. Dazu soll es möglichen sein, dass parallel mehrere Instanzen der Testskripte gestartet werden können, um einen Mehrbenutzerbetrieb zu simulieren.
- Mit dem Werkzeug sollen Zeitmessungen während der Ausführung der Testskripte auf dem Testobjekt durchgeführt werden können. Das Zeitverhalten des Testobjektes soll erfasst werden, um später, nach dem Sammeln von Erfahrungswerten, Ausreißer erkennen zu können.

1.2 Bachelorarbeit im Unternehmen

Im Rahmen des Bachelorabschlusses ist diese Arbeit in Kooperation mit dem Unternehmen PASCAL Beratungsgesellschaft für Datenverarbeitung m.b.H. und der Hochschule für Angewandte Wissenschaften Hamburg entstanden. Alle in dieser Arbeit aufgeführten Geschäftsprozesse, Arbeitsabläufe, Codeausschnitte u. ä. sind aus datenschutzrechtlichen Gründen verfremdet, beziehen sich aber auf Verfahren des Unternehmens.

PASCAL Beratungsgesellschaft für Datenverarbeitung m.b.H.
Mörkenstraße 5, 22767 Hamburg
Telefon: +49 (0) 40 / 30 611 - 0
Internet: https://www.pascal.de/

Die Firma PASCAL stellt ihren Kunden das ERP-System INTEGRA® bereit und bietet hierfür ein Customizing und Consulting an. Das ERP-System INTEGRA® kann, dank seiner baukastenartigen Architektur, modifiziert werden.

Der Grundbaukasten für das ERP-System INTEGRA® stammt aus der Unternehmenskooperation ORGA-SOFT® Cooperation. Mit diesem Grundbaukasten erstellt die Firma PASCAL für ihre Kunden aus unterschiedlichen Branchen maßgeschneiderte Lösungen.

1.3 Lösungsidee

Für die Umsetzung zur Einführung des Werkzeuges hat die Firma PASCAL ein Pilotprojekt gestartet. Eine Werkzeugevaluation und die Auswahl des Werkzeuges fand vorher statt. Ausgewählt wurde das Werkzeug AutoIt [Aut18].

Gegenstand dieser Bachelorarbeit ist es, die von der Firma PASCAL umgesetzte Einführung des Werkzeuges mit einem Verfahren zur Werkzeugeinführung aus der Literatur als Rahmenvorlage zu beschreiben und zu analysieren. Dieses Verfahren zur Werkzeugeinführung stammt aus [WSLR14] und besteht aus den fünf Phasen:

Phase 1 – Grundsätzliche Entscheidung über den Einsatz eines Werkzeuges

Phase 2 – Festlegung von Anforderungen

Phase 3 – Evaluation von Werkzeugen

Phase 4 – Auswertung der Evaluation und Auswahl

Phase 5 – Einführung (Pilotprojekt)

Die Phasen 1 und 2 des Verfahrens dienen zur Festlegung von Zielen und Anforderungen, also der Vorbereitung der Evaluation. Die Evaluation entfällt, da diese vorher stattfand. Die dabei durchgeführte Nutzwertanalyse ist im Anhang A.1 zu finden. Die in der Evaluation (Phase 3) betrachteten Alternativen TestComplete [Tes18] und der JitBit Macro Recorder [Sof18] konnten sich gegenüber AutoIt nicht durchsetzen. Die Gründe die zur Auswahl (Phase 4) von AutoIt führten, werden in 4.5 aufgeführt. Die vorher genannten Punkte werden in Kapitel 4 beschrieben.

Mit der Phase 5 des Verfahrens erfolgt die Beschreibung und Analyse des von der Firma PASCAL durchgeführten Pilotprojektes. Dazu wird die Durchführung des Pilotprojektes in Kapitel 5 beschrieben und in Kapitel 6 ausgewertet und analysiert. Teil dieser Auswertung ist die Analyse der zuvor gesetzten Ziele, Kosten und Anforderungen. Außerdem erfolgt eine

Auseinandersetzung mit den Herausforderungen während des Pilotprojektes. Abschließend erfolgt eine Aufführung von Verbesserungsvorschlägen zur Programmierung der Testskripte, die den Aufwand der Erstellung und der Wartung der Testskripte reduzieren.

Die Zielsetzungen der Bachelorarbeit sind:

- die gesetzten Ziele, Kosten, Risiken und Anforderungen an das Werkzeug zu prüfen und zu analysieren
- die im Pilotprojekt aufgetretenen Herausforderungen zu dokumentieren und zu analysieren
- und etwaige Vorschläge zur Verbesserung in Form von „Best Practices“ aufzuzeigen.

1.4 Abgrenzungen

Ein Vergleich oder eine Evaluation verschiedener Werkzeuge wird in dieser Arbeit nicht durchgeführt. Diese fand vorher statt. Eine Nutzwertanalyse dazu ist im Anhang A.1 zu finden. Die Gründe, warum AutoIt als Werkzeug gewählt wurde, sind in 4.5 aufgeführt.

In dieser Arbeit wird ein Bericht und eine Analyse über die Werkzeugeinführung mithilfe des Verfahrens zur Werkzeugeinführung aus [WSLR14] durchgeführt.

Mit Automatisierung ist in dieser Arbeit die Automatisierung der Testdurchführung und nicht der Testfall- oder der Testdatengenerierung gemeint. Eine automatisierte Testauswertung wird in dieser Arbeit ebenfalls nicht betrachtet. Der Tester muss selber bewerten, ob der Test erfolgreich war.

Die Arbeit und die Ergebnisse der Arbeit beziehen sich auf das ERP-System INTEGRA® der Firma PASCAL und gelten nicht im Allgemeinen für ERP-Systeme.

1.5 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut:

> **Kap. 1**: Die **Einleitung** beinhaltet die Problemstellung und skizziert die Lösungsidee. Eine Abgrenzung der nicht behandelten Aspekte ist aufgeführt.

Kap. 2: Das Kapitel **„theoretischer Hintergrund“** betrachtet den theoretischen Hintergrund des Testens mit speziellem Fokus auf den Systemtest am Testobjekt ERP-System und dessen Charakteristika.

Kap. 3: Das Kapitel **„betrieblicher Hintergrund“** stellt den Projekthintergrund vor, also: die Firma PASCAL, das Testobjekt ERP-System INTEGRA® und das Testwerkzeug AutoIt.

Kap. 4: In Kapitel 4 wird der **Lösungsansatz** vorgestellt. Dazu werden die Zielsetzungen, Kosten, Risiken und Anforderungen anhand des Verfahrens aus [WSLR14] beschrieben. Es wird auf die Evaluation von Werkzeugen mit Nutzwertanalyse im Anhang A.1 verwiesen. Es wird begründet, warum AutoIt gewählt wurde.

Kap. 5: In Kapitel 5 findet die **Durchführung** des Lösungsansatzes statt. Dabei werden die Inhalte des Pilotprojekts beschrieben.

Kap. 6: In Kapitel 6 erfolgt die **Auswertung** des Pilotprojektes. Insbesondere die Zielsetzungen, Kosten, Risiken und Anforderungen werden gegen die Vorgaben aus Kapitel 4 geprüft. Außerdem werden Herausforderungen des Pilotprojektes aufgeführt.

Kap. 7: In Kapitel 7 **„Zusammenfassung und Ausblick“** erfolgt eine Zusammenfassung der erreichten Ziele und eine Betrachtung der noch zu klärenden Aspekte für die Zukunft.

2 Theoretischer Hintergrund

In diesem Kapitel wird der theoretische Hintergrund mit speziellem Fokus auf den Systemtest am Testobjekt ERP-System betrachtet.

2.1 Allgemeine Grundlagen

Anhand des Titels der Arbeit „Automatisierung von Standardtestfällen der Anwendungslogik eines ERP-Systems“ lassen sich folgende thematische Schwerpunkte identifizieren:

Titel	**Thema**
Automatisierung von	**Testautomatisierung**
Standardtestfällen der Anwendungslogik	**Systemtest** als Teststufe
eines ERP-Systems	System Under Test: **ERP**

Tabelle 2.1: Theoretischer Hintergrund – Schwerpunkte

Die Schwerpunktthemen Automatisierung und ERP-System leiten sich anhand des Titels ab. Die Einordnung in die Stufe des Systemtests erfolgt aufgrund der Formulierung „Standardtestfälle der Anwendungslogik“. Gemeint sind hier geschäftsprozessbasierte Testfälle. Getestet werden soll das System von außen, ähnlich einer Blackbox. Diese Schwerpunktthemen sind Gegenstand der folgenden theoretischen Hintergrundbetrachtung.

2.1.1 Einordnung der Teststufe Systemtest

Im Allgemeinen lassen sich bei Softwareprojekten unterschiedliche technische Abstraktionsebenen festmachen. Bezogen auf Teststufen sind das beim allgemeinen V-Modell:

→ Abnahmetest, Systemtest, Integrationstest, Komponententest

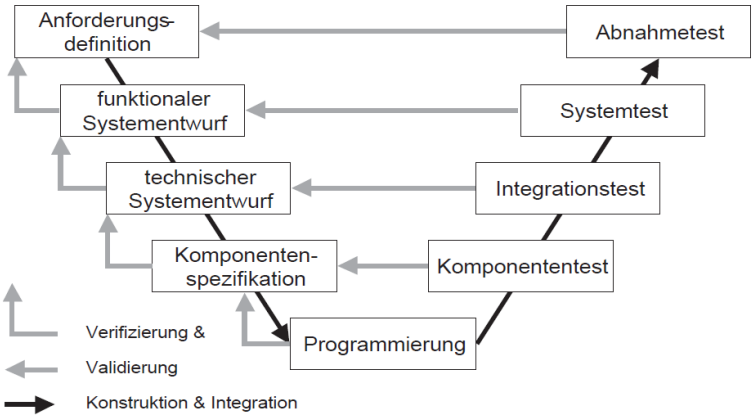

Abbildung 2.1: V-Modell aus [LS12]

Die wichtigsten Kennzeichen des V-Modells sind [LS12]:

- Konstruktions- und Testaktivitäten sind getrennt, aber gleichwertig (linke, rechte Seite).
- Das „V“ veranschaulicht die Prüfaspekte Verifizierung und Validierung.
- Es werden arbeitsteilige Teststufen unterschieden, wobei jede Stufe „gegen“ ihre korrespondierende Entwicklungsstufe testet.
- Die jeweiligen Teststufen sind technisch sehr unterschiedlich und erfordern deshalb unterschiedliche Testmethoden, Testwerkzeuge und spezialisiertes Testpersonal.
- Proaktives Testen: Testplanung, -analyse und -design sind zwar nicht dargestellt, werden aber impliziert und finden parallel zu den Entwicklungsschritten der linken Seite statt.

Eine explizite Darstellung der Parallelität von Entwicklung und Test [LS12] durch das sogenannte W-Modell ist bspw. in [WSLR14] zu finden.

Eine sehr wichtige Erkenntnis der unterschiedlichen technischen Abstraktionsebenen ist, dass Fehler am einfachsten auf derselben Abstraktionsstufe gefunden werden, auf der sie entstanden sind [LS12].

Mit Blick auf Abbildung 2.1 folgt oberhalb der Programmierung die Teststufe Komponententest. In dieser Stufe wird der Programmiercode auf Komponentenebene getestet. Generell sind die

Testobjekte hier Klassen, Methoden oder Funktionen. Auf der nächsten Stufe – Integrationstest – werden zusammenhängende, integrierte Teile in ihrer korrekten Zusammenwirkung getestet, bspw. die Datenzugriffsschicht in Zusammenarbeit mit einer Datenbank. Die Stufe des Systemtests folgt als nächstes. Hier wird das gesamte System zum ersten Mal nach voriger Integration getestet. Die abschließende Stufe ist die des Abnahmetests. Der Test auf dieser Stufe kann dem des Systemtests sehr ähnlich oder gleich sein. Die Unterschiede sind der Verantwortungsbereich und die Testumgebung. Der Kunde ist hier verantwortlich und getestet wird in dessen Umgebung. [LS12]

2.1.2 Systemtest

Der Systemtest (vgl. Abbildung 2.1) dient der Prüfung, ob das System als Ganzes die spezifizierten Anforderungen erfüllt [LS12].

Als Gründe, warum der Systemtest wichtig ist und nicht ausbleiben sollte, nennen [LS12]:

- Die Perspektive gegenüber den vorigen Teststufen ist nun nicht mehr technisch und aus Sicht des Softwareherstellers. Die Perspektive erfolgt nun aus Sicht des Kunden. Es ist ein Blick auf das System als Ganzes [LS12].
- [Fow13] nennt hier ebenfalls als Vorteil, dass die Anwendung im Ganzen getestet wird und so Fehler in der Interaktion zwischen Teilkomponenten der Anwendung getestet werden können. Mit Komponententests ist das nicht möglich.
- „Viele Funktionen und Systemeigenschaften resultieren aus dem Ineinandergreifen aller Systemkomponenten und sind somit erst auf Ebene des Gesamtsystems beobachtbar und testbar.“ [LS12]

Im Systemtest wird das System als Ganzes betrachtet, und zwar in einer Testumgebung, die der späteren Produktivumgebung möglichst nahe kommt [LS12], [Lüb18].

Der Systemtest erfordert eine separate Testumgebung. Die Tester haben keine oder nur geringe Kontrolle über Parameter und Konfiguration der Produktivumgebung. Durch den gleichzeitig zum Test weiterlaufenden Betrieb der anderen Kundensysteme werden die Randbedingungen des Tests unter Umständen schleichend verändert. Die durchgeführten Systemtests sind schwer oder nicht mehr reproduzierbar [LS12].

Ziele des Systemtests sind [LS12]:

- die gestellten Anforderungen (funktionale und nicht funktionale) zu prüfen
- Fehler und Mängel aufgrund falsch, unvollständig oder im System widersprüchlich umgesetzter Anforderungen aufzudecken.

[LS12] identifizieren als Probleme beim Systemtest:

- **Unklare Kundenanforderungen:** Die Kundenanforderungen sind nicht dokumentiert, sondern nur „in den Köpfen“ einiger am Projekt beteiligter Personen vorhanden.
- **Versäumte Entscheidungen:** Müssen die ursprünglichen Anforderungen nachträglich identifiziert werden, wird es vorkommen, dass zu einem Sachverhalt unterschiedliche Ansichten und Vorstellungen bei den beteiligten Personen existieren. Dieses Zusammentragen der Informationen ist sehr zeit- und kostenintensiv.
- **Projekte scheitern:** Wenn Anforderungen nicht dokumentiert sind, fehlen den Entwicklern klare Ziele. Die Wahrscheinlichkeit, dass das konstruierte System die impliziten Kundenanforderungen erfüllt, ist deshalb außerordentlich gering.

Zum Verständnis soll an dieser Stelle verdeutlicht werden, welche Fragestellungen auf dieser Ebene im Kontext von ERP möglich sind:

- Werden die Bestellungen überhaupt erfasst? → Also erscheint die Bestellung nach Erfassung im System (Wird die Bestellung in der Datenbank gespeichert?)?
- Wird die Bestellung auch korrekt verbucht? → Werden im ERP-System dafür Eingangsbelege erzeugt und angezeigt?
- Wird die Bestellung nach der Erfassung auch abgefertigt? → Werden Ausgangsbelege wie Lieferscheine erzeugt?
- Wird der Kunde über den Bestellstatus informiert? → Werden Informationen über den Bestellstatus an den Kunden verschickt?

2.1.3 Funktionaler Test

Das funktionale Testen ist das Testen des Systems oder von Komponenten basierend auf ihren funktionalen Spezifikationen [GTBe17]. Mit Blick auf das allgemeine V-Modell resultieren diese funktionalen Spezifikationen aus den Anforderungsdefinitionen, die auf den Anforderungen der Kunden basieren. Das Testen beim Systemtest gestaltet sich hier wie das Testen einer Black-Box. Man hat keinen internen Zugriff, sondern testet das System als Ganzes von außen, gesteuert über Schnittstellen, die von außen verfügbar sind [LS12].

Eine Auswahl an Testmethoden aus [LS12] sind hier u. a.:

- Anforderungsbasiertes Testen – „Ein Ansatz zum Testen, der auf den Anforderungen basiert. Aus ihnen werden die Testziele und Testbedingungen abgeleitet. Dazu gehören Tests, die einzelne Funktionen tätigen oder solche, die nicht funktionale Eigenschaften wie Zuverlässigkeit oder Gebrauchstauglichkeit untersuchen.“ [GTBe17]
- Anwendungsfallbasiertes Testen – „Verfahren, bei dem Testfälle so entworfen werden, dass damit Szenarien der Anwendungsfälle durchgeführt werden.“ [GTBe17]
- Geschäftsprozessbasiertes Testen – „Testverfahren, bei dem die Testfälle auf Grundlage von Beschreibungen und/ oder der Kenntnis von Geschäftsprozessen hergeleitet und ausgewählt werden.“ [GTBe17]

Zur Verfeinerung bzw. systematischen Abdeckung von Eingaben und Ermittlung von Testfällen, können weitere Black-Box-Testentwurfsverfahren eingesetzt werden:

- Äquivalenzklassenbildung – Einordnung der Eingaben in Wertebereiche. Auswahl einzelner Repräsentanten aus den Wertebereichen [LS12].
- Grenzwertanalyse – Eingaben am Rande der Äquivalenzklassen [GTBe17].
- Ursache-Wirkungs-Graph-Analyse – Entwurf von Testfällen mit Ursache-Wirkungs-Graphen, einer graphischen Darstellung zur Organisation und Darstellung der Zusammenhänge verschiedener möglicher Ursachen eines Problems [GTBe17].
- Entscheidungstabellentechnik – Entwurf von Testfällen in Entscheidungstabellen mit Regeln, die jeweils aus einer Kombination von Bedingungen und den dazugehörigen Aktionen bestehen [GTBe17].

2.1.4 Nicht-funktionaler Test

Der nicht funktionale Test basiert auf den Attributen der nicht funktionalen Anforderungen. Einige Merkmale sind hier laut [LS12]: Zuverlässigkeit, Benutzbarkeit, Effizienz. Bei nicht funktionalen Tests wird grundsätzlich untersucht „wie gut“ bzw. mit welcher Qualität das System seine Funktion erbringt [LS12].

Folgende nicht funktionale Systemeigenschaften in entsprechenden Tests sind u. a. laut [LS12]:

- „Lasttest: Messung des Systemverhaltens in Abhängigkeit steigender Systemlast (z. B. Anzahl parallel arbeitender Anwender, Anzahl Transaktionen).“ [LS12]
- „Performanztest: Messung der Verarbeitungsgeschwindigkeit bzw. Antwortzeit für bestimmte Anwendungsfälle, in der Regel in Abhängigkeit steigender Last.“ [LS12]
- „Stresstest: Beobachtung des Systemverhaltens bei Überlastung.“ [LS12]
- „Test der Stabilität/ Zuverlässigkeit im Dauerbetrieb (z. B. Ausfälle pro Betriebsstunde bei gegebenem Benutzungsprofil).“ [LS12]
- „Test auf Robustheit gegenüber Fehlbedienung, Fehlprogrammierung, Hardwareausfall usw. sowie Prüfung der Fehlerbehandlung und des Wiederanlaufverhaltens (recovery).“ [LS12]
- „Prüfung auf Änderbarkeit/ Wartbarkeit: Verständlichkeit und Aktualität der Entwicklungsdokumente; modulare Systemstruktur usw.“ [LS12]

2.1.5 Test nach Änderungen und Regressionstest

Nach Fertigstellung der Hauptentwicklung folgt bei Softwaresystemen generell die Softwarewartung und Weiterentwicklung, die sogenannte inkrementelle Entwicklung. Typischerweise folgt nach den Iterationszyklen der inkrementellen Entwicklung ein änderungsbezogener Test, auch Regressionstest genannt. [LS12]

„Sowohl durch Fehlerkorrektur- bzw. Wartungsarbeiten als auch bei der Weiterentwicklung werden Teile vorhandener Software geändert oder neue Softwarebausteine ergänzt. Hier muss durch geeignete Wiederholung von Tests nachgewiesen werden, dass die vormaligen Fehler wie

beabsichtigt tatsächlich korrigiert wurden (Fehlernachtest), aber auch, dass die Änderungen keine unbeabsichtigten Seiteneffekte haben (Regressionstests).“ [LS12]

„Der Regressionstest ist ein erneuter Test eines bereits getesteten Programms nach dessen Modifikation mit dem Ziel, nachzuweisen, dass durch die vorgenommenen Änderungen keine neuen Defekte eingebaut oder freigelegt wurden. Damit Testfälle für einen Regressionstest verwendet werden können, müssen sie wiederholbar, d. h. ausreichend dokumentiert sein. Testfälle, die in Regressionstests benötigt werden, sind bevorzugte Kandidaten für die Testautomatisierung.“ [LS12]

[LS12] liefern eine Auswahl von Regressionstestfällen speziell für den Systemtest:

- Wiederholung nur von denjenigen Tests aus dem Testplan, denen hohe Priorität zugeordnet ist.
- Bei funktionalen Tests Verzicht auf gewisse Varianten (Sonderfälle).
- Einschränkung der Tests auf bestimmte Konfigurationen (z. B. nur Test der englischsprachigen Produktversion, nur Test auf einer bestimmten Betriebssystemversion usw.).
- Einschränkung der Tests auf bestimmte Teilsysteme oder Teststufen.

2.1.6 Testautomatisierung

[GTBe17] definiert Testautomatisierung als: „Einsatz von Softwarewerkzeugen zur Durchführung oder Unterstützung von Testaktivitäten, z. B. Testentwurf, Testausführung und Soll-/Ist-Vergleich.“

Als wesentlichen Grund zur Effizienzsteigerung von Tests, um deren hohe Kosten zu reduzieren, identifizieren [SBS12] die Testautomation.

Auf die Frage, wie man Kosten für das Testen sonst reduzieren könnte, stellen sie ([SBS12]) fest:

- Weniger testen: Beschränkung auf die Teile im System, die im Zweifel am meisten Schaden anrichten. → unterm Strich nicht ausreichend

- Effizienter testen: Vollständige Automatisierung des Systemtests → von Planung bis Auswertung – von Anfang bis Ende

Eine Liste der grundlegenden Aktivitäten hierbei sind [SBS12]:

- Automatische Ableitung der logischen Testfälle aus der Anforderungsdokumentation
- Automatisierte Erzeugung eines Testplans
- Automatische Erstellung eines Testentwurfs
- Automatische Generierung der Testdaten
- Automatisierte Erzeugung physikalischer Testfälle
- Automatische Generierung der Testprozeduren
- Automatische Instrumentierung des Codes
- Die automatische Testausführung
- Die automatische Ergebnisprüfung
- Automatische Kontrolle der Testüberdeckung
- Automatisch generierte Testmetrik

Als die wichtigsten Voraussetzungen für die Testautomation identifizieren [SBS12]:

- Die Formalisierung der Anforderungsspezifikation
- Die Standardisierung der Testdokumente
- Die Definition der Datenwertebereiche
- Die Qualifizierung der Tester

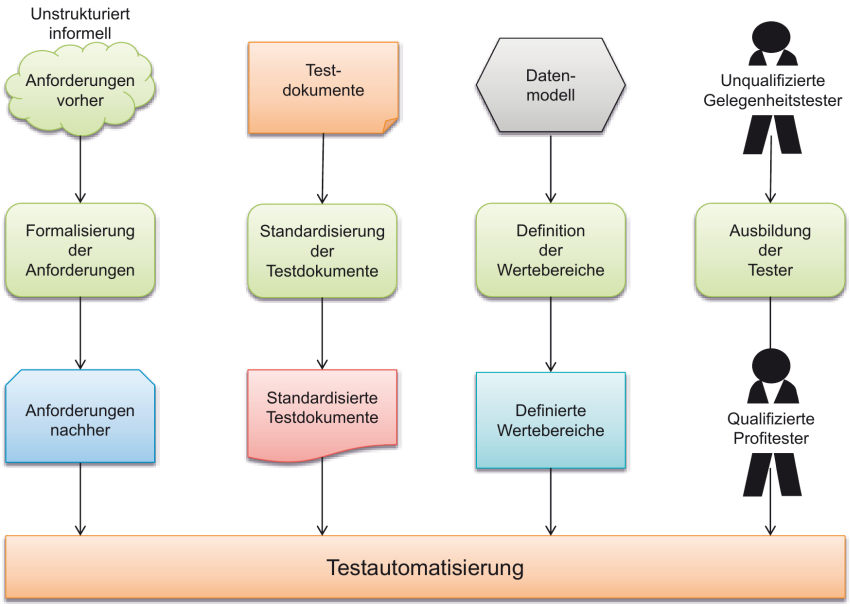

Abbildung 2.2: Voraussetzung der Testautomation aus [SBS12]

[SBS12] diskutieren zwei Alternativen zur Software-Testproblematik:

1. **Weniger Testen:** Durchführung einiger Stichproben, um festzustellen, ob das System überhaupt lauffähig ist. Beschränkung auf einige Kernfunktionen → langfristig die teuerste Alternative, denn Fehler sind in Software immer vorhanden. Fehler, die in Produktion oder Wartung auftreten, sind aufwendig zu korrigieren und teuer. Dazu kommen Ausfallkosten bei Kunden. Selbst kleine Fehler können bis zu einen Personentag kosten. Verbleiben hundert solcher Fehler in der Software, kosten diese hundert Personentage. Testwirtschaftlichkeit beginnt mit der Eindämmung der Fehlerkosten [SBS12].

2. **Massiver Personaleinsatz:** Die Effizienz hängt hier am Faktor Mensch. Testaufgaben erfordern ein gewisses Maß an Konzentration, die Menschen generell nur über eine gewisse Zeit aufrecht erhalten können. Lässt die Konzentration nach, werden Menschen nachlässig und machen Fehler. Automaten bzw. Maschinen ermüden nicht. Die Wahrscheinlichkeit, einen Fehler aufzudecken, ist jederzeit gleich groß. Personalaufstockung für manuelles Testen ist nach [SBS12] ebenfalls eine zu teure Alternative.

2.1.7 Typen von Testwerkzeugen im Systemtest

Wird als Testschnittstelle direkt die grafische Oberfläche des Softwaresystems verwendet, können sogenannte Capture & Replay Tools (C&R) [LS12] oder Skriptsprachen zum Einsatz kommen. Erstere funktionieren nach dem Prinzip eines Videorekorders. Zunächst werden Benutzereingaben auf dem Testobjekt aufgezeichnet und anschließend zur Wiederausführung gebracht. Die Aufzeichnungen können nach der Aufnahme editiert werden. [BF12] unterscheiden hier drei Kategorien von Capture & Replay Tools:

- Solche der ersten Generation, die koordinatenbasiert arbeiten.
- Solche der zweiten Generation, die sich in die Anwendung einhängen und Zugriff auf die Komponenten, z. B. IDs, Labels etc. bekommen; sogenannte komponentenbasierte Tools.
- Solche der dritten Generation, die mit Bilderkennung arbeiten.

Alternativ kann eine Skriptsprache zum Einsatz kommen, mit der die grafische Oberfläche angesteuert wird. Notwendig wird dann ein sogenanntes Spy-Tool, das Fenster und Elemente der grafischen Oberfläche (engl. Controls), wie Textfelder und Buttons, identifiziert.

Herausforderungen mit C&R-Tools Die vorigen Herangehensweisen klingen sehr verlockend, weil sie einfach umsetzbar scheinen. Doch gibt es hier einige Schwierigkeiten zu beachten [LS12]:

- Koordinatenbasierte Tools erzeugen keine robusten Aufnahmen. Ändert sich die Auflösung, schlägt die Wiedergabe sehr wahrscheinlich fehl, weil Aktionen beim Abspielen an falscher Stelle ausgeführt werden.
- Tools mit Komponentenbasierter Erkennung haben z. B. das Problem der Objektidentifizierung. Oft sind Komponenten derart auf Kundenbedürfnisse angepasst, dass diese nicht identifiziert werden können.
- Dann bleibt oft nur der Griff zu Tools, die mit Bilderkennung arbeiten. Allerdings kann es hier Probleme geben, wenn sich bspw. das Design des Testobjekts ändert.

Mischformen der vorgenannten C&R-Werkzeugtypen versuchen der Problematik entgegenzuwirken. Allerdings wird ein generelles Problem dieser Testweise auf der Stufe des Systemtests deutlich. Man ist der ganzen „Unsicherheit“ der Ebene des Systems ausgesetzt.

Häufige Herausforderungen bei der Objektidentifizierung sind:

- Das Auffinden der Elemente an sich. [San14]
- Die Dauer des Auffindens der Elemente. [San14]
- Scheinbar zufällig werden Elemente nicht gefunden; beim erneuten Versuch diese Auffzufinden aber doch. Vergleiche hierzu [San14].

Was die Robustheit anbelangt, steht man bei der Automatisierung von Oberflächentests also vor Herausforderungen.

Als Alternative setzen [San14] hier bspw. auf eine maßgeschneiderte Eigenlösung.

Eine weitere Alternative ist es, Testfälle aufzuteilen in kleinere Teile, um diese unabhängig voneinander zu machen, ähnlich wie in [Fow11].

2.1.8 Auswahl und Einführung von Testwerkzeugen

Das hier geschilderte Verfahren zur Auswahl und Einführung von Testwerkzeugen basiert auf dem Verfahren aus [WSLR14]. An dieser Stelle soll das Verfahren skizziert werden. Im praktischen Teil der Arbeit dient es als wissenschaftliche Vergleichsgrundlage gegenüber der intuitiv gewählten Herangehensweise der Firma PASCAL.

Ein wesentlicher Bestandteil des Verfahrens ist die Überlegung [WSLR14], ob ein Open-Source-, ob ein kommerzielles Werkzeug ausgewählt oder ob eine maßgeschneiderte Eigenentwicklung durchgeführt werden soll.

Eine systematische Werkzeugauswahl erfolgt unabhängig von der Herkunft in folgenden Phasen [WSLR14]:

P1 Grundsätzliche Entscheidung über den Einsatz eines Werkzeugs

- Identifikation von Zielen und Nutzen

- Betrachtung möglicher Alternativlösungen
- Betrachtung der Kosten
- Identifikation und Management von Risiken

P2 Festlegung von Anforderungen

- Funktionale Anforderungen an das Testwerkzeug
- Nicht-funktionale Anforderungen an das Testwerkzeug
- Anforderungen an Begleitleistungen (z. B. telefonischer Produktsupport und elektronische Dokumentation) zum Produkt
- Aufstellung eines Kriterienkatalogs

P3 Evaluation

- Selektion der Evaluationskandidaten
- Planung und Setup
- Bewertung der Werkzeuge anhand der Kriterien
- Berichterstellung

P4 Auswertung und Auswahl des zu beschaffenden Werkzeugs

- Bewertungsverdichtung und Entscheidungsvorbereitung
- Entscheidung

P5 Einführung des ausgewählten Werkzeugs

- Werkzeuge und notwendige Prozessreife
- Pilotprojekt
- Wichtige Rollen im Pilotprojekt
- Ergebnisauswertung und Entscheidung

- Verbreitung

Nach Abschluss der Einführung und Inbetriebnahme des Werkzeuges folgt der Betrieb. Für diesen weiteren Verlauf sollten zur Sicherstellung der Langlebigkeit des Werkzeuges entsprechende Maßnahmen, wie ein interner Support und die Weiterentwicklung des Werkzeuges, vorbereitet werden. Eine mögliche Außerbetriebnahme des Werkzeuges ist einzuplanen.

2.2 ERP-System

In diesem Abschnitt wird die Art der Anwendung des Testobjekts (System Under Test) vorgestellt. Dazu werden zunächst Charakteristika von ERP aufgeführt. Danach wird im Allgemeinen beschrieben, was unter ERP verstanden wird.

2.2.1 Charakteristika von ERP-Systemen

Auf Grundlage der Ausführungen in [KRG00] lassen sich folgende Charakteristika von ERP-Systemen aufstellen:

- **ERP-Software ist Standardsoftware:** Diese muss vor der Inbetriebnahme in gewissem Maße auf die spezifischen Anforderungen des jeweiligen Unternehmens zugeschnitten werden. [KRG00], [LLS15]
- **Individualisierung (Customizing):** Der Prozess der Software-Individualisierung wird als Customizing bezeichnet. Sie ermöglicht eine individuelle Konfiguration der ERP-Implementierungen. Ein reichhaltiges Konfigurationspotenzial von ERP-Software ergibt sich aus dem Bereich bereits vorkonfigurierter Alternativen und der Anzahl alternativer Prozesse und Transaktionen. [KRG00], [LLS15]
- **ERP-Systeme nutzen eine gemeinsame integrierte Datenbank:** ERP-Software basiert auf einer zugrunde liegenden integrierten Datenbank, die Stamm- und Transaktionsdaten konsistent speichert. Hierdurch ist eine bereichsübergreifende Datennutzung möglich, ohne dass die Daten mehrfach eingegeben und gepflegt werden müssen. [KRG00], [AM17]
- **ERP-Software ist Anwendungssoftware:** ERP unterstützt u. a. mehr als 12 Module. Die wichtigsten, die ein ERP-System unterstützt, sind: Marketing, Vertrieb, Unternehmenslösung, Produktionsplanung, Qualitätsmanagement, Anlagenbuchhaltung, Materi-

alwirtschaft, Kostenkontrolle, Personalwesen, Projektmanagement, Finanzen, Anlagenwartung. [KRG00], [DFFS13]

- **ERP-Software fokussiert auf sich wiederholende Geschäftsprozesse:** Obwohl Komponenten der wichtigsten ERP-Lösungen auf der obersten Ebene in verschiedenen funktionalen Modulen wie Finanzbuchhaltung oder Vertrieb organisiert sind, verfolgen sie alle eine prozessorientierte Sicht auf Unternehmen. Unterstützt werden wohlbekannte Geschäftsprozesse wie Beschaffung, Auftragsabwicklung oder Zahlungsprozesse. [KRG00], [LLS15]

- **ERP-Systeme integrieren Geschäftsprozesse über Funktionen hinaus:** Typische Geschäftsprozesse werden nahtlos über Funktionen hinweg unterstützt, so dass der Benutzer oft nicht erkennt, in welchem Funktionsmodul er tatsächlich arbeitet. [KRG00], [LLS15]

- **ERP-Software bündelt Funktionen in Modulen** und **ERP-Systeme bestehen aus einer Menge integrierter Module:** Die Anwendungsmodule von ERP sind über die unterstützten Funktionen und die beteiligten Daten integriert. Diese Module, wie zum Beispiel Personalwesen, Vertrieb, Finanzen und Produktion, ermöglichen eine organisationsübergreifende Integration von Informationen durch eingebettete Geschäftsprozesse. [KRG00], [NTD12]

- **ERP-Systeme haben eine einheitliche Benutzeroberfläche:** Ein weiteres technisches Merkmal von ERP-Software ist neben den integrierten Anwendungen und Daten die konsistente grafische Benutzeroberfläche (GUI) über alle Anwendungsbereiche hinweg. Somit nimmt ein Benutzer die ERP-Lösung als eine einzelne Anwendung wahr, unabhängig von dem Modul, mit dem er arbeitet. [KRG00], [AM17]

- **ERP-Systeme sollten international einsetzbar sein:** Sie sollten mehrere Währungen und verschiedene Länderanforderungen unterstützen. [KRG00]

2.2.2 Was ist ERP?

ERP steht für Enterprise-Resource-Planning und bedeutet auf Deutsch übersetzt in etwa Unternehmensressourcenplanung. Eine einfache Erklärung nur anhand des Namens könnte folgendermaßen lauten: ERP ist die Planung von Ressourcen im unternehmerischen Kontext.

Wobei mit Ressourcen generell solche wie Kapital, Personal, Betriebsmittel, Material usw. gemeint sind.

Ein Blick in die Literatur liefert weitere Erkenntnisse zum Begriff ERP:

[LLS15] definieren ERP-System so: „ERP-Systeme sind integrierte unternehmensweite Anwendungssysteme, die zur Koordination wichtiger interner Prozesse eines Unternehmens dienen."

[KRG00] definieren ERP als eine allumfassende gebündelte Softwarelösung, die die gesamte Bandbreite der Geschäftsprozesse und -funktionen eines Unternehmens integrieren soll, um aus einer einzigen Informations- und IT-Architektur einen ganzheitlichen Blick auf das Unternehmen bekommen zu können.

[DFFS13] definieren ERP als ein Konstrukt, das alle Geschäftsprozesse und Daten eines Unternehmens in eine umfassende integrierte Struktur bringt.

Durch [KRG00] und [LLS15] wird eine andere Bedeutung deutlich. Bei ERP geht es um die Abbildung und Integration von Geschäftsprozessen in Unternehmen. [DFFS13] bringen zusätzlich noch den Aspekt der Datenhaltung mit ein.

[AM17] stellen sogar fest, dass der Begriff ERP unglücklich gewählt ist, da die meisten ERP-Softwareprodukte weder die Ressourcen noch deren Planung, sondern vielmehr die integrative Abwicklung von Geschäftsprozessen in den Mittelpunkt stellen.

[Kee15] kommen zu einer ähnlichen Feststellung. Anstelle des ERP-Begriffs müsste richtigerweise die Bezeichnung Enterprise-Resource-Management (ERM-)System verwendet werden, da diese den Sachverhalt besser trifft: „Unter Enterprise-Resource-Management (ERM) versteht man das Management aller materiellen und/ oder immateriellen Ressourcen in einer planvoll organisierten Wirtschaftseinheit, um Leistungen und Güter herzustellen und/ oder abzusetzen, im Sinne von Zielsetzung, Planung, Entscheidung, Ausführung, Kontrolle, Information und Koordination." [Kee15]

[KRG00] stellen drei verschiedene Ausführungen von ERP-Systemen vor:

1. **Generische Ausführung (Standardsoftware):** Diese Ausführung zielt auf eine Reihe von Branchen ab und muss vor dem Einsatz konfiguriert werden.

2. **Vorkonfiguriert:** Eine bereits vorkonfigurierte Ausführung basierend auf der generischen Standardausführung. Vorkonfiguriert wird für Industrien mit wohlbekannten Geschäftsprozessen und -funktionen.

3. **Individuell angepasste Ausführung:** Eine individuelle Anpassung der generischen oder vorkonfigurierten Ausführung an die jeweiligen Firmenanforderungen.

Traditionelle ERP-Systeme basieren auf einer Drei-Schichten-Architektur, bei der Datenbank, die Anwendungen und die Präsentation drei logisch unabhängige Schichten bilden [KRG00].

[Bay13] nennt zwei modernere Architekturen, die heute vor allem zum Einsatz kommen:

- Hub and Spoke: Zentral organisierte Hub-and-Spoke-Architekturen, die auf Middleware-Konzepten basieren [Bay13].

- Service-orientierte Architekturen (SOA): Mit Schnittstellen, die auf Standards wie Web-Services basieren. Die Funktionalitäten werden bei dieser Architektur in neutral gekapselten Services zur Verfügung gestellt. Diese lassen sich in verschiedenen Anwendungszusammenhängen einsetzen und wiederverwenden [Bay13].

2.2.3 Zusammenfassung

Zusammengefasst kann man ERP-Systeme also als die Integration einer Sammlung von Geschäftsprozessen auf einer zentralen Datenbank beschreiben. Dieser prinzipielle Aufbau von ERP-Systemen ist in folgender Abbildung dargestellt:

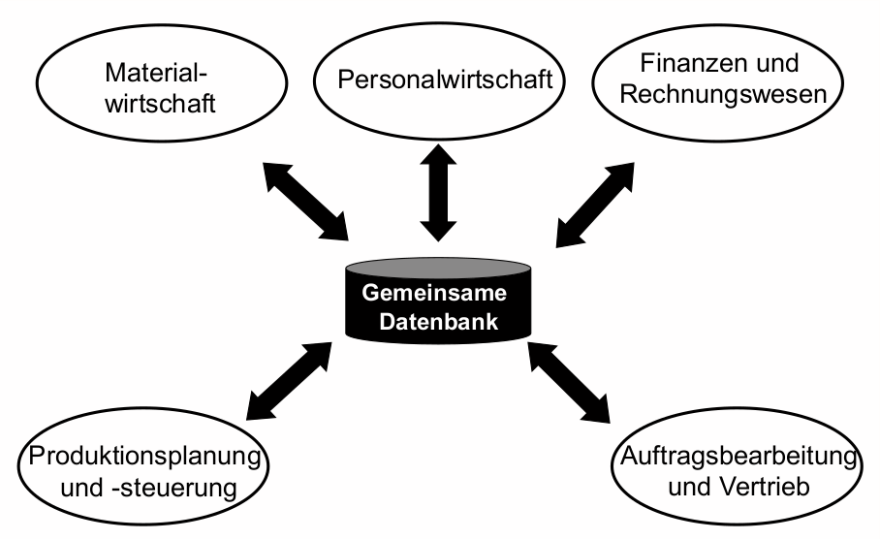

Abbildung 2.3: Prinzipieller Aufbau eines ERP-Systems aus [AM17]

3 Betrieblicher Hintergrund

In diesem Kapitel werden die Produktionsumgebung der Firma PASCAL und das Testwerkzeug AutoIt vorgestellt.

3.1 Produktionsumgebung

In diesem Abschnitt werden das ERP-System INTEGRA®, der Entwicklungsprozess und die Entwicklungsumgebung der Firma PASCAL vorgestellt, um den Hintergrund zu verdeutlichen.

3.1.1 INTEGRA® – die Grundversion

Wie in 1.2 Bachelorarbeit im Unternehmen beschrieben, stammt das ERP-System INTEGRA® aus einer Kooperation von Unternehmen. Die Grundversion von INTEGRA® wird dabei von der Kooperation ORGA-SOFT® Cooperation geliefert.

Diese Grundversion bietet Module mit Funktionalitäten aus den Bereichen Material-/ Warenwirtschaft, Rechnungswesen, BI/ Controlling/ DMS, Vertrieb und Personalwesen an.

Werden von den beteiligten Firmen der Kooperation bei deren spezialisierten Eigenentwicklungen Lösungen geschaffen, die für alle Beteiligten interessant sein können, werden diese nach Prüfung in den Standard übernommen.

Die Firma PASCAL bietet ihren Kunden das ERP-System INTEGRA® an. Im Rahmen dieser Bachelorarbeit wird als exemplarisches Beispiel der Bereich Fruchthandel vorgestellt.

Um Kunden aus dem Bereich des Fruchthandels INTEGRA® zur Verfügung stellen zu können, musste der vorkonfigurierte Standard auf die Gegebenheiten der Branche im Fruchthandel zugeschnitten werden. Erreicht wird das durch Customizing.

Das Customizing und die wesentlichen Besonderheiten im Bereich des Fruchthandels lassen sich durch erhöhte Lagerumschlagshäufigkeit und QS-Anforderungen erklären:

- Viele Früchte weisen eine schnelle Verderblichkeit auf und können deshalb nicht lange gelagert werden.
- Früchte sind Lebensmittel und haben aufgrund gesetzlicher Bestimmungen eigene Qualitätssicherungsanforderungen. Oftmals sind mehrere Zertifikate notwendig.

Grundsätzlich wird bei neuen Kunden geprüft, ob deren Geschäftsprozesse in INTEGRA® abbildbar sind und wie hoch der Customizing-Aufwand ist. In dieser Phase steht die Beratungsleistung von PASCAL im Vordergrund.

3.1.2 Entwicklungsprozess und -umgebung der Firma PASCAL

Zur Unterstützung der Entwicklung von INTEGRA® betreibt die Firma PASCAL eine eigene Ausführung von INTEGRA®. Konkret handelt es sich hier um ein Ticketsystem, das folgende Funktionalitäten bietet:

- Tickets
- Offene Arbeiten
- Zeiterfassung
- Abrechnungen

Das Vorgehen bei der Entwicklung des ERP-Systems INTEGRA® geschieht nach dem Wasserfallmodell mit Rückführungen. Sind die Entwicklungsarbeiten abgeschlossen, erfolgen manuelle Tests auf technischer und funktionaler Ebene basierend auf den Anforderungsdokumenten. Die Tests erfolgen gemeinschaftlich mit den Projektleitern. Sind diese abgeschlossen, wird der Kunde informiert. Dieser muss nach seiner erfolgreichen Abnahme die Freigabe erteilen. Ausgeliefert werden größere Teile oder Pakete von Anpassungen in Form von Rollouts. Dringende Änderungen werden per Hotfix eingespielt. Alle bei diesen vorkommenden Schritten erfassten Dokumente werden den jeweiligen offenen Arbeiten oder Servicetickets zugeordnet.

Das ERP-System wird in der Programmiersprache Object Pascal und der Entwicklungsumgebung Delphi, primär für Windows-Umgebungen, entwickelt. Die Entwicklung bei PASCAL findet auf verschiedenen Terminal-Servern in einer Citrix-Umgebung statt. Auf den Terminal-

Servern laufen die Windows-Server-Versionen Microsoft Windows Server 2008 R2, 2012 R2, 2016. Subversion (SVN) [Sub18] wird für die Versionsverwaltung eingesetzt.

Grundsätzlich existieren pro Kunde drei Systeme:

- **Programmiersystem:** Entwicklung.
- **Testsystem:** Einspielen von Anpassungen, Fehlerbehebungen. Tests und Abnahme durch Kunden.
- **Echtsystem:** Das Produktivsystem des Kunden.

Der Kunde hat Zugriff auf das Test- und das Echtsystem.

Es existiert ein Build-Server, auf dem die Programmiersysteme nächtlich kompiliert werden. Die Testsysteme werden bei Bedarf aktualisiert.

Das Erstellen der Echtsysteme geschieht folgendermaßen:

- Per Hotfix: Nach Vereinbarung und Abnahme durch den Kunden.
- Rollout: Sammlung von offenen Arbeiten nach Absprache und Abnahme durch den Kunden.

Die Systeme sind entweder direkt beim Kunden oder in der Cloud von PASCAL installiert. Der Zugriff für Kunden auf diese Cloud erfolgt über Citrix – eine Remote-Verbindung.

INTEGRA® hat eine Drei-Schichten-Architektur, bestehend aus:

- Datenbankschicht
- Geschäftslogik (Domänen, Rollen etc.)
- Repräsentationsschicht

Die Datenschicht besteht aus mindestens drei Datenbanken:

- Mandanten – Mandantenspezifische Daten: Stammdaten, Bewegungsdaten, Belege, Artikel

- Standard – Allgemeingültige Daten: Benutzerkennungen, Drucklayouts, Rechtesteuerung
- System – Logische Steuerung der Daten: Metamodell/ Systemdictionary, Übersetzungen

Das Metamodell/ Systemdictionary ist eine Abstraktionsebene in der System-Datenbank. Mit dieser zusätzlichen Ebene wird das Anpassen von Elementen der grafischen Oberfläche ohne Programmierung möglich.

Eine Replikation der mandantenspezifischen Datenschicht ist möglich, falls ein Kunde mehrere Firmen oder mehrere Standorte besitzt.

Eine Besonderheit stellt die System-Datenbank dar. In ihr wird das sogenannte Metamodell gespeichert. Mit diesem Metamodell wird das Customizing – also die Konfiguration des ERP-Systems – ermöglicht. Als Beispiel können hier pro Kunde Felder unterschiedlich konfiguriert und übersetzt werden. In der vorkonfigurierten Version für den Fruchthandel könnten bspw. zwei Kunden mit unterschiedlichen Artikelnummern behandelt werden. Kunde A hat nur numerische Artikelnummern. Kunde B hat alphanumerische Artikelnummern. Das Metamodell überwacht und steuert hier diese Unterscheidung.

3.2 Das Werkzeug AutoIt

In diesem Abschnitt erfolgt eine allgemeine Beschreibung des Werkzeuges AutoIt. Beschrieben werden außerdem wichtige Hilfswerkzeuge im Lieferumfang von AutoIt. Dazu zählen das Spy-Tool, der Editor und der Skript-Compiler von AutoIt.

3.2.1 AutoIt Allgemein

AutoIt v3 ist eine BASIC-ähnliche Skriptsprache, die zur Automatisierung der Windows-GUI und zur Erstellung von allgemeinen Skripts entwickelt wurde. AutoIt bietet Funktionen zur Simulation von Tastenanschlägen, Mausbewegungen und Fenster-/ Steuermanipulationen. AutoIt ist klein, „self-contained“ (in sich geschlossen) und läuft auf allen Windows-Versionen „out-of-the-box“. AutoIt ist „Freeware“, also kostenlos, aber „Closed Source“ [Aut18].

AutoIt wurde ursprünglich für den Rollout von vielen Computer-Installationen entwickelt, um diese zuverlässig zu automatisieren und zu konfigurieren. Im Laufe der Zeit hat es sich zu

einer mächtigen Sprache entwickelt, die komplexe Ausdrücke, UDFs (User Defined Functions, dt. benutzerdefinierte Funktionen), Schleifen und alles andere unterstützt, was ein erfahrener Skripter erwarten würde [Aut18].

Eine Auswahl der Funktionalitäten von AutoIt, entnommen von der Internetseite [Aut18]:

- Einfach zu erlernende BASIC-ähnliche Syntax
- Simulation von Tastenanschlägen und Mausbewegungen und -aktionen
- Interaktion mit allen Standard-Windows-Steuerelementen (Standard Windows Controls)
- Skripte können in eigenständige ausführbare Dateien kompiliert werden
- Detaillierte Hilfe und große Community-basierte Support-Foren
- Kompatibel mit: Win XP, 2003, Vista, 2008, Windows 7, 2008 R2, Windows 8, 2012 R2
- AutoIt muss nicht installiert werden und ist in einer portablen Version verfügbar

AutoIt wurde so konzipiert, dass es so klein wie möglich und eigenständig ist, ohne dass externe DLL-Dateien oder Registrierungseinträge erforderlich sind, um die Verwendung auf Servern zu ermöglichen. Skripte können mit Aut2Exe in eigenständige ausführbare Dateien kompiliert werden.

3.2.2 Features

AutoIt kommt mit einer Reihe von Tools. Hier werden die wichtigen aufgeführt, die bei der Erstellung der Skripte notwendig sind oder unterstützen.

3.2.2.1 Au3Info – AutoIt-Spy-Tool

Mit dem AutoIt-Spy (Au3Info) lassen sich bei der Erstellung von Skripten zur Ausführung der grafischen Oberfläche (GUI) eines Testobjektes, Information zu Steuerungselementen der GUI ermitteln. Das Tool lässt sich separat auf Dateiebene oder aus dem Skripteditor heraus starten.

Um Information zu einem Steuerungselement zu erhalten, zieht man per Drag-&-Drop ein Fadenkreuz des AutoIt-Spy auf das gewünschte Steuerungselement. Der Spy liefert dann

entsprechende Informationen, die zur Identifikation der Steuerungselemente genutzt werden können, bspw. Fenstertitel oder Name der Control usw.

Eine Abbildung des AutoIt-Spy-Tools **Au3Info** ist im Anhang A.2 zu finden.

3.2.2.2 Editor – SciTE4AutoIt3

Der Editor basiert auf dem freien Editor SciTE [Sci18]. Für AutoIt existiert die angepasste Version SciTE4AutoIt3 [Aut18]. Grundlegende Features sind: Syntax Highlighting, Autovervollständigung von Code und Anzeige der Dokumentation. Letztere sind out-of-the-box aber nur für die mitgelieferten Funktionen verfügbar. Die mitgelieferten Tools lassen sich über das Kontextmenü von SciTE oder über Tastenkombinationen starten.

Skripte werden entweder direkt aus Editor heraus zur Ausführung gebracht oder können in eine ausführbare Exe-Datei kompiliert werden.

Eine Abbildung des Editors **SciTE4AutoIt3** ist im Anhang A.3 zu finden.

3.2.2.3 AutoIt Skript Compiler Aut2Exe

Mit dem AutoIt Skript Compiler Aut2Exe lassen sich Skripte in ausführbare Exe-Dateien kompilieren. Diese sind eigenständig und haben keine Abhängigkeiten [Aut18]. Aus dem Editor lässt sich dafür per Menü oder Tastenkombination ein Dialog starten. In diesem können Einstellungen zur Skriptkompilierung vorgenommen werden. Die Voreinstellungen reichen aber generell aus:

Eine Abbildung des AutoIt Skript Compiler Dialogs **Aut2Exe** ist im Anhang A.4 zu finden.

3.2.2.4 Elemente der Skriptsprache AutoIt

Die wichtigsten grundlegenden Elemente der Skriptsprache AutoIt sind:

- **Bedingte Anweisung und Verzweigung:** If...Then...Else, Select...Case, Switch...Case, Ternärer Operator
- **Schleifen:** For...Next, While...WEnd, Do...Until, For...In...Next
- **Benutzer-definierbare Funktionen,** um eigene Funktionalitäten in Funktionen kapseln zu können.

- **Präprozessor Direktiven:** Als wichtigste Direktive ist #include zu nennen. Diese dient zum Einfügen von Source-Dateien. Nicht vorhanden sind Direktiven, wie: ifdef, ifndef.

4 Lösungsansatz

In diesem Kapitel wird der Lösungsansatz vorgestellt. Dazu wird das Verfahren zur Werkzeugeinführung aus [WSLR14] als Rahmenvorlage verwendet.

4.1 Vorgehen

In diesem Abschnitt soll das Vorgehen anhand des in 1.3 Lösungsidee erwähnten Verfahrens erfolgen. Hauptbetrachtungsgegenstand in dieser Arbeit ist dabei die Pilotprojekt-Phase – Phase 5 des Verfahrens. Die Phasen 1 bis 4 des Verfahrens dienen der Vorbereitung zur Auswahl eines Werkzeuges. Für diese Arbeit stand das Werkzeug AutoIt aber schon vorher fest. Das Ergebnis dieser ersten Phasen ist in Form einer Nutzwertanalyse im Anhang A.1 Evaluation Werkzeuge zu finden. Die Phase 5 des Verfahrens zur Betrachtung des Pilotprojektes erfolgt in Kapitel 5 Durchführung des Pilotprojekts.

Die ersten Phasen sollen hier dennoch behandelt werden, um den Zusammenhang zu verdeutlichen und zu zeigen warum das Werkzeug AutoIt verwendet wurde. Insbesondere die Zielsetzungen, Anforderungen, Kostenaspekte und Herausforderungen werden aufgeführt. Am Ende der Pilotprojektphase soll eine Auswertung der Zielsetzungen, Anforderungen, Kostenaspekte und Herausforderungen und der neuen Herausforderungen, die während des Pilotprojektes auftraten, erfolgen. Diese Auswertung, mit etwaigen Verbesserungsvorschlägen, wird in das nächste Kapitel 6 Auswertung ausgelagert.

Als Grundlage für das Pilotprojekt dient der Geschäftsprozess eines Kunden aus der Branche des Fruchthandels. Der Geschäftsprozess wird in Phase 5 beschrieben. Das Dokument dieses Geschäftsprozesses ist die Testfallspezifikation des Geschäftsprozesses in Excel. Der Geschäftsprozess wird zusammen mit dem Kunden in natürlicher Sprache in einer Excel-Datei tabellarisch dokumentiert. Anhand der Testfallspezifikation, bestehend aus Prozessschritten, Prüfpunkten, Eingabedaten und Sollwerten, wird die Kernfunktionalität eines Standardtestfalles mit dem Werkzeug AutoIt und den zur Verfügung stehenden weiteren Testmitteln erstellt. Diese Testmittel sind:

- Das Werkzeug AutoIt und die Hilfswerkzeuge 3.2 Das Werkzeug AutoIt.
- Eine AutoIt-Datei mit vorbereiteten Funktionen, die sinnvolle Funktionalität kapselt, wie das Lesen aus Ini-Dateien zum Steuern des Testskriptes.

In den nachfolgenden Abschnitten erfolgen nun die einzelnen Phasen des Verfahrens. Die ersten Phasen dienen der Auswahl der Werkzeuges. Das Pilotprojekt beginnt ab Phase 5.

4.2 Phase 1 – Grundsätzliche Entscheidung über den Einsatz eines Werkzeugs

In diesem Abschnitt werden die Ziele, Kosten und Risiken der Werkzeugeinführung betrachtet.

4.2.1 Identifikation von Zielen

Basierend auf den Schilderungen aus der Problemstellung, ergeben sich folgende Ziele:

Ziel 1: Verbesserung der Qualität des ERP-Systems, um frühzeitig Fehlerwirkungen auf der Systemebene entdecken zu können, durch Einführung von Testautomation mit einem Werkzeug, um diese exakter zu machen.

Ziel 2: Verbesserung der vorhandenen manuellen Testprozesse auf funktionaler Systemebene, um diese kostengünstiger, schneller und exakter zu machen.

Ziel 3: Das Werkzeug soll von der Erlernbarkeit so sein, dass ein QS-Consultant es nach einer Einführung bedienen kann.

Ziel 4: Das Werkzeug soll den Ersteller in die Lage versetzen, derart robuste Testakten erstellen zu können, die im Anschluss des nächtlichen Build-Vorgangs als Regressionstest laufen.

Ziel 5: Bei Bedarf sollen Entwickler oder Projektleiter selbst eigene Testakten erstellen können, um nach Programmierarbeiten zu prüfen.

Ziel 6: Getestet werden soll, wie ein Anwender die Anwendung bedient. Es sollen komplexe Standardtestfälle (Kernfunktionalität) von Kunden erstellt werden können und diese sollen robust sein.

Ziel 7: Es soll Last-Messungen durch Simulieren eines Mehrbenutzerbetriebes ermöglichen.

Ziel 8: Das Werkzeug soll Zeit-Messungen ermöglichen.

Ziel 9: Wirtschaftlichkeitsaspekte sollen betrachtet werden: Erste Erfahrungswerte, zwecks Dauer der Erstellung der Testakte durch den Tester, sollen beim Pilotprojekt gesammelt werden.

Ziel 10: Die Ausführung der grafischen Oberfläche mit einem Werkzeug ist gewollt. Die Effekte, die sich dabei störend auf die Robustheit auswirken, sind gewollt.

4.2.2 Betrachtung der Kosten

Es existieren noch keine Erfahrungswerte. Ein maximales Zeitbudget von 500 Stunden wurde als Obergrenze abgeschätzt. In diesem Zeitrahmen ist das Pilotprojekt von allen Beteiligten zu erfüllen.

Die Kosten, verursacht durch den QS-Consultant als Tester/ Testskript-Ersteller, beziehen sich auf:

- Weitere Anforderungseinholung für die Testfallspezifikation durch den QS-Consultant bei Kunden oder erfahrenen Mitarbeitern.
- Erstellung des Skripts durch den QS-Consultant.
- Verstehen der Testakte also des Geschäftsprozesses, um diesen überhaupt in ein Skript übersetzen zu können.
- Erlernen der weiteren Testmittel:
  - des Werkzeuges
  - des Testrahmens

Weitere Kosten beziehen sich auf Kosten durch die Erstellung eines für den QS-Consultant geeigneten Testrahmens. Funktionalitäten dafür wurden in einer separaten Datei mit Testfunktionen gesammelt. Erstellt wurden diese von einem Entwickler.

Wiederkehrende Kosten entstehen durch die Wartung und Erweiterung der Testmittel. Ebenfalls durch den Entwickler:

- Testfallspezifikationen

- Testrahmen
- Testfunktionen
- Schulungen und Hilfestellungen für den QS-Consultant

Für die spätere Inbetriebnahme des Werkzeuges werden folgende Kosten anfallen:

- Einrichtung des Werkzeuges
- Schulung der Mitarbeiter
- Erstellen von Handbüchern und Hilfen

4.2.3 Identifikation und Management von Risiken

Folgende mögliche Risiken werden bei der Erstellung der Testfälle eingeschätzt:

- Kann mit dem Werkzeug überhaupt ein komplexer Geschäftsprozess in INTEGRA® umgesetzt werden?
- Kann der QS-Consultant eine Testakte im zeitlichen Budget umsetzen, bzw. kann ein Mitarbeiter mit diesen Fähigkeiten effizient so einen Testfall umsetzen?

Als weiteres Risiko wird ein hoher Wartungsaufwand der Testmittel identifiziert. Bei den Testmitteln handelt es sich um:

- die Testfallspezifikation des Geschäftsprozesses in Excel
- die Umsetzung der Testfallspezifikation mit dem Werkzeug

Da bewusst ein Werkzeug zur Automatisierung der Oberfläche von Anwendungen eingesetzt werden soll, lassen sich folgende technische Bereiche, die Herausforderungen verursachen könnten, identifizieren:

- Die Identifizierung der Steuerelemente des Testobjektes.
- Herausforderungen beim Timing bei der Interaktion mit dem Testobjekt.

- Die Robustheit der Testmittel, die später die Ausführung des Testfalls durchführen.

4.3 Phase 2 – Festlegung von Anforderungen

Ziel der Festlegung von Anforderungen ist eigentlich die Erstellung eines Kriterienkatalogs für eine darauf folgende Evaluation von Werkzeugen durch eine Nutzwertanalyse. Da das Werkzeug für diese Arbeit vorgegeben wurde, erfolgt hier nur noch die Darstellung der funktionalen und nicht funktionalen Anforderungen an das Werkzeug. Das Ergebnis der Evaluation ist im Anhang A.1 Evaluation Werkzeuge zu finden.

Als Beteiligte werden identifiziert:

- Die Firma mit wirtschaftlichen Interessen
- Die Abteilungen mit ihren Kundenprojekten
- Die Anwender:
  - QS-Consultant: Erstellung von Testskripten mit Kernfunktionalität für das Pilotprojekt und die spätere Betriebsphase.
  - Entwickler, Projektleiter: Erstellung von Testskripten, aber keine ganzen Geschäftsprozesse, sondern nur für den Teil, der in ihren Entwicklungsbereich fällt.

Funktionale Anforderungen sind im Speziellen bei der Identifizierung von Steuerelementen, der Robustheit und der Integration von AutoIt in die bestehende Umgebung von hoher Bedeutung.

Aus Sicht der Firma PASCAL:

- Wegen der Ersteinführung des Werkzeuges ist die Erfassung von Zeiten interessant (für das Pilotprojekt wurde ein maximales Zeitbudget von 500 Stunden eingerichtet):
  - Erfassung der Zeiten zur Umsetzung durch den QS-Consultant
    - Das bedeutet, das Werkzeug muss durch Personal ohne technischen Hintergrund im Maximal-Zeitbudget zum Ziel führen und erlernbar sein
  - Bereitstellung eines adäquaten Testrahmens für den QS-Consultant durch einen Entwickler

  - Schulung des QS-Consultant wg.:
    * technischer Fragen zum Umgang mit dem Werkzeug durch einen Entwickler
    * Verständnisfragen zum Geschäftsprozess an erfahrene Mitarbeiter

Aus Sicht der Abteilungen:

- Das Werkzeug muss sich in die vorhandene Infrastruktur integrieren lassen:
  - Installationen auf Kundensystemen sind nicht erwünscht
  - Das Werkzeug muss sich in das ERP integrieren lassen, um damit verteilt werden zu können.
- Die Kernfunktionalität im Sinne von Standardtestfällen in Form von Geschäftsprozessen in INTEGRA® soll sichergestellt werden
  - Die Auswahl der Eingaben und Prozessschritte erfolgt in Zusammenarbeit mit dem Kunden
  - Tiefergreifende Spezifikation von Eingaben durch andere Spezifikationstechniken, wie Äquivalenzklassenbildung oder Entscheidungstabellentechnik werden nicht betrachtet, da sie den Rahmen sprengen würden. Die Geschäftsprozesse sind dafür zu komplex. Für Tests im kleineren Rahmen durch Entwickler werden die Spezifikationstechniken aber für die Zukunft in Betracht gezogen.
- Aus Gründen der Effektivität:
  - Das Tool soll auch deshalb von einem QS-Consultant erlernbar sein, um dessen Einfachheit bei der Erlernbarkeit zu bestätigen.
    * Das bedeutet, das Werkzeug muss durch Personal ohne technischen Hintergrund im Maximal-Zeitbudget zum Ziel führen und erlernbar sein
  - Das Tool darf wegen des vorher Geschilderten keinen großen Entwicklungsaufwand erzeugen.
  - Das Tool darf wegen des vorher Geschilderten keinen großen Wartungsaufwand erzeugen.

- Es sollen keine Tools mit rein bildbasierter Erkennung verwendet werden.
- Das Werkzeug soll Lasttests und Zeitmessungen ermöglichen.
  - Lasttests sollen ermöglicht werden: Dazu sollen mehrere Instanzen des Werkzeuges parallel ausgeführt werden können, um den Parallelbetrieb simulieren zu können.
  - Es sollen Zeitmessungen bei der Ausführung der Geschäftprozesse durch das Werkzeug erfolgen.

Aus Sicht des QS-Consultants:

- Das Tool muss von einem QS-Consultant erlernt werden können
- Dafür müssen Hilfestellungen existieren:
  - Support durch Entwickler, die ERP-System und Werkzeug kennen und erklären können.
  - Support für die Geschäftsprozesse des ERP-Systems muss existieren.
  - Handbücher und weitere Hilfsdokumente müssen bereitgestellt werden.
- Es muss ein Testrahmen für den QS-Consultant geschaffen werden, mit dem der Mitarbeiter umgehen kann.
- In Frage kommen dabei Werkzeuge wie Capture & Replay und rudimentäre Skriptsprachen.
- Lösungen, die tiefgreifende Erfahrungen eines Entwicklers notwendig machen, werden nicht betrachtet. Z. B. solche, bei denen Programmierkonzepte, wie Objektorientierung, vorausgesetzt werden.
- Werkzeuge, die nur einfache technische Konzepte von Skriptsprachen umsetzen, sind akzeptabel. Bspw. wie es bei AutoIt der Fall ist.

Aus Sicht der Entwickler, Projektleiter:

- Grundsätzlich gelten die Anforderungen des QS-Consultants.

- Das Tool soll sich schnell erlernen lassen, was das Pilotprojekt mit der Rolle des QS-Consultants bestätigen soll.
- Das Werkzeug soll so weit vorbereitet sein, dass ein Rahmen mit Best Practices existiert, so dass es sich ohne Einarbeitungsaufwand einsetzen lässt.

4.4 Phase 3 – Evaluation von Werkzeugen

Die Evaluation von Werkzeugen fand schon vor dieser Arbeit statt. Das Ergebnis dieser Evaluation ist im Anhang A.1 Evaluation Werkzeuge zu finden. Die Schritte beim Vorgehen dieses Verfahrens sind in 2.1.8 Auswahl und Einführung von Testwerkzeugen aufgezählt und werden in [WSLR14] näher beschrieben.

4.5 Phase 4 – Auswertung und Auswahl des zu beschaffenden Werkzeugs

Als Ergebnis der Evaluation (siehe Anhang A.1 Evaluation Werkzeuge) wurde AutoIt als Werkzeug gewählt. Die in der Evaluation betrachteten Alternativen TestComplete [Tes18] und der JitBit Macro Recorder [Sof18] konnten sich gegenüber AutoIt nicht durchsetzen.

Nachfolgend ist aufgelistet, warum das Werkzeug AutoIt ausgewählt wurde:

- AutoIt muss nicht installiert werden.
- AutoIt ist „Self Contained“ und hat unter Windows keine Abhängigkeiten und ist „out-of-the-box“ lauffähig.
- Die Skripte sind einfache Textdateien und nicht in einem proprietären Dateiformat.
- Eine Testakte lässt sich in eine ausführbare Exe-Datei kompilieren und so auf einfache Weise verteilen.
- Es gibt Mitarbeiter im Unternehmen, die Erfahrungen mit dem Skripten und im Umgang mit AutoIt haben.
- Die vorgenannten Gründe machen die Integration in dem vorhanden Entwicklungsprozess mit AutoIt sehr einfach. Mit AutoIt kann ein Testfall bspw. in eine ausführbare Datei

kompiliert werden. Diese Datei kann dann einfach auf ein Kundensystem kopiert und ausgeführt werden.

- AutoIt ist kostenlos: Stand Juni 2018.

Generell überzeugt AutoIt durch die Einfachheit. Demgegenüber steht allerdings der hohe Aufwand zur Erstellung eines Testrahmens. AutoIt ist eine reine Skriptsprache. TestComplete als reines Werkzeug liefert vieles schon mit, z. B. Logging und Reporting und Metriken hierfür.

5 Durchführung des Pilotprojekts

In diesem Teil wird die Pilotprojekt-Phase (Phase 5 des Verfahrens aus [WSLR14]) mit dem Werkzeug AutoIt beschrieben. Die nachfolgenden Abschnitt umfassen:

- Wichtige Rollen im Pilotprojekt
- Vorbereitungen für das Pilotprojekt
- Beschreibung des Testrahmens
- Das Testszenario: Geschäftsprozess eines Kunden aus dem Fruchthandel
- Vorgehen beim Skripten mit AutoIt

Die während des Pilotprojektes aufgetretenen Herausforderungen und Auswertungen erfolgen in Kapitel 6 Auswertung.

5.1 Wichtige Rollen im Pilotprojekt

Die folgenden Rollen sind beim Vorhaben vertreten:

- **Projektleiter**: Führt Aufgaben des Testmanagement, wie Planung, Organisation, Dokumentation, aus.
- **Wissenslieferanten zur Testfallspezifikation**: Generell soll der Kunde die Testfallspezifikation liefern. Zur Unterstützung stehen erfahrene Mitarbeiter der Firma PASCAL bei Verständnisproblemen bei der Übersetzung ins Testskript zur Verfügung.
- **Entwickler/ Werkzeugverantwortlicher:** Übernimmt die Entwicklung von Testfunktionen für den Testrahmen in AutoIt. Gibt Hilfestellung bei Fragen zu technischen Aspekten und Erstellen der Testskripte.

- **Tester:** Der Tester im Pilotprojekt ist ein QS-Consultant. Bei Bedarf erweitert er die Testfallspezifikation und erstellt basierend auf ihr das Testskript.

5.2 Vorbereitungen für das Pilotprojekt

Zur Vorbereitung für das Pilotprojekt wurde folgende Infrastruktur erschaffen bzw. erweitert:

1. Es wurde im internen Projektmanagementsystem ein Eintrag für das Pilotprojekt angelegt, um alle zeitlichen Aufwände darin buchen zu können.

2. Es wurde mit dem Versionierungssystem SVN ein eigenes Repository für das Pilotprojekt angelegt.

3. Ein Testrahmen mit Testfunktionen wurde von Entwicklern angefertigt und soweit vorbereitet, dass der QS-Consultant beginnen kann, Testskripte zu erstellen. Dabei nutzt er nachfolgende Testmittel, um ein Testskript in der AutoIt-Skript-Sprache zu erstellen:

    - Testfallspezifikation in Excel

    - Testfunktionen in einer AutoIt-Skript-Datei

    - Spy-Tool zur Identifizierung von Merkmalen von Steuerelementen

4. Zur Unterstützung wurde eine Schnellstartanleitung im firmeneigenen Wiki angelegt.

5. Die Entwickler/ Werkzeugverantwortlichen, die an der Entwicklung des Testrahmens beteiligt waren, geben bei Bedarf Hilfestellung bei Erstellung der Skripte.

6. Die Testfallspezifikation basiert auf den geschäftsprozessbasierten Testfällen des Kunden und wird von diesem geliefert.

Die Nummern in Abbildung 5.1 entsprechen denen der obigen Aufzählung aus Punkt 5.2 Vorbereitungen für das Pilotprojekt.

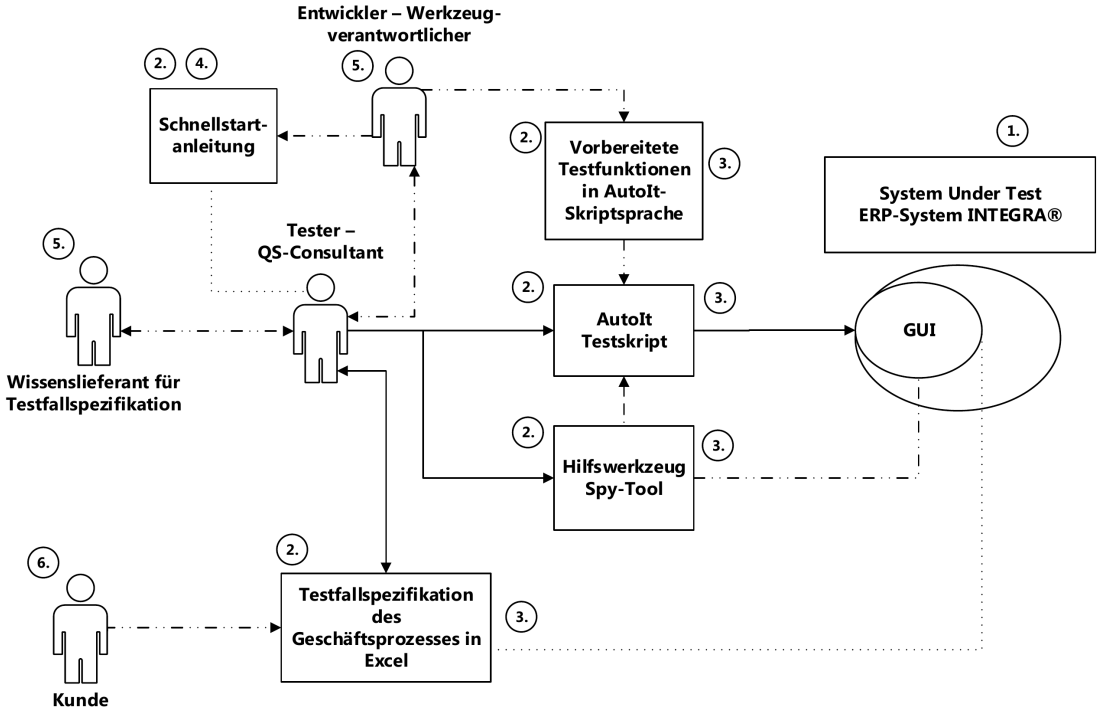

Abbildung 5.1: Eigene Darstellung der Test-Infrastruktur

Die zur Verfügung stehenden Testmittel sind gemäß Abbildung 5.1:

- Die Testfallspezifikation des Geschäftsprozesses in Excel (oder anderes Tabellenkalkulationsformat). Diese wurde in Zusammenarbeit mit dem Kunden angefertigt. Basierend auf dieser wird durch den Tester ein Skript in AutoIt erstellt.
- Das Werkzeug AutoIt mit den Hilfswerkzeugen, erwähnt in 3.2 Das Werkzeug AutoIt.
- Die vorbereiteten Testfunktionen in einer AutoIt-Skript-Datei.
- Das AutoIt-Testskript selbst. Parametrisiert wird das Testskript mit einer Ini-Datei (siehe Abbildung 5.2).
- Die Schnellstartanleitung für den Testrahmen im firmeneigenen Wiki.

5.3 Beschreibung des Testrahmens

Für den Tester wurde zur Anfertigung der Skripte ein Testrahmen durch den Werkzeugverantwortlichen angelegt. Dazu wurde eine Datei angelegt, in der Funktionalitäten gekapselt wurden, die das Anlegen des Testskriptes vereinfachen sollen. Konkret handelt es sich dabei um Funktionen, die ein gewisses Maß an Aktionen zur Interaktion mit der Oberfläche des Testobjektes durchführen. Beispielsweise das Starten des ERP-Systems in Form der Desktop-Anwendung. Außerdem wurden einige Funktionen von AutoIt erweitert, bspw. das Warten und Aktivieren eines Fensters in einer Funktion.

Das eigentliche Testskript wird dann in einer eigenen oder mehreren Dateien (hierbei handelt es sich um einfache Textdateien mit der Endung .au3) erzeugt. Zuvor wurde durch den Werkzeugverantwortlichen ein Beispiel-Projekt als Vorlage angelegt und eine Hilfe im Firmen-Wiki erzeugt. Die vorbereitete Datei mit den Testfunktionen wird per Anweisung inkludiert.

Für die Parametrisierung des Skriptes wurde eine Ini-Datei angelegt. In der Ini-Datei sind Konfigurationseinstellungen abgelegt. Die Ini-Datei dient zum Steuern des Skripts durch die Eingabewerte und dient als Container für Ausgabewerte, die für anschließende Skripte wieder als Eingabe dienen.

Der Testrahmen besteht aus den Dokumenten in folgender Abbildung

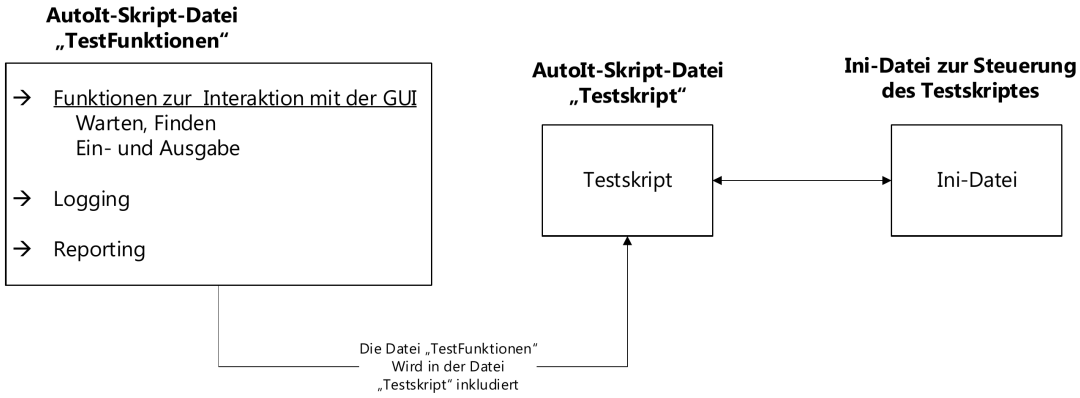

Abbildung 5.2: Eigene Darstellung der Testrahmens

Die Datei „TestFunktionen“ enthält folgende Funktionen:

- Logging in Text-Datei.
- Reporting in Excel-Datei.
- Hotkeys für das Starten, Stoppen und Pausieren der Ausführung.
- Eigene Funktionen, die im Sinne eines Endanwenders mit der grafischen Oberfläche interagieren, z. B. zum Starten und Beenden von INTEGRA®.
- Workaround-Funktionen, insbesondere für den Umgang mit Grids.
- Funktionen bzw. Erweiterungen bestehender Funktionen für die Interaktion mit der GUI:
  - Warten auf Fenster mit Timeout.
  - Warten auf Controls mit Timeout.
  - Senden von Tastaturbefehlen mit Timeout.

5.4 Das Testszenario: Geschäftsprozess eines Kunden aus dem Fruchthandel

Der exemplarische Testfall basiert auf einem Geschäftsprozess eines Kunden aus dem Bereich des Fruchthandels.

Der Geschäftsprozess lautet:

> Grundsätzlicher Durchlauf von Partieerstellung über Verkaufs- und Einkaufs-Kosten bis Partieabrechnung.

Der gesamte Prozess besteht aus den vier Teilprozessen:

1. Partieerstellung aus Manifest (Warenavis: Ankündigung einer Warenlieferung)
2. Verkauf - Erfassung
3. Einkaufsrechnungen
4. Partieabrechnung

Eine Partie ist die Sendung eines Lieferanten auf einem Schiff.

Der gesamte Prozess wird in folgender Abbildung vereinfacht dargestellt und anschließend erläutert:

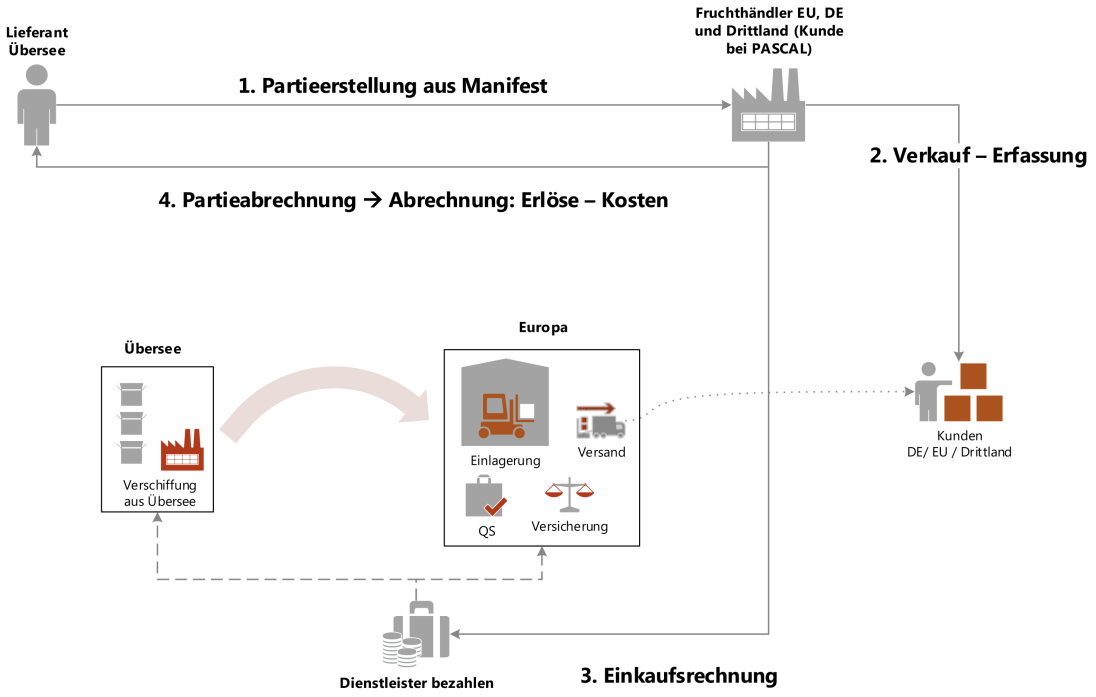

Abbildung 5.3: Eigene grafische Darstellung des obigen Geschäftsprozesses

Der Geschäftsprozess basiert auf einem Vertrag zwischen dem Fruchthändler (Kunde bei PASCAL) und dem Lieferanten in Übersee. Gegenstand des Vertrages ist die Abmachung, dass der Fruchthändler die gelieferten Waren zum „Best Price“ (bester Verkaufspreis auf dem Markt) verkauft.

Unabhängig von den kaufmännischen Geschäftsprozessen wird parallel laufend Ware von Übersee nach Europa verschifft. In Europa wird die Ware von vom Fruchthändler beauftragten Dienstleistern weiterverarbeitet. Bspw. werden einige der Früchte erst eingelagert, weil diese erst reifen müssen. Andere Früchte werden sofort in den Versand zu den Kunden des Fruchthändlers gebracht. Weitere Dienstleistungen sind u. a. Qualitätssicherungsmaßnahmen

zur Überprüfung der Warenqualität und der Abschluss von Versicherungen, um gegen den Verderb und Diebstahl von Waren abgesichert zu sein.

Der Geschäftsprozess beginnt mit dem Teilprozess **„1. Partieerstellung aus Manifest“**. Dazu übersendet der Lieferant dem Fruchthändler eine Manifestdatei. Diese Datei beinhaltet ein Warenavis: eine Ankündigung eines Warenzuganges.

Im nächsten Prozess **„2. Verkauf - Erfassung“** erfolgt der Verkauf der Waren an die Kunden des Zielmarkts. Kunden kommen hier bspw. aus der Branche des Lebensmitteleinzelhandels.

Die daraus generierten Umsätze des Fruchthändlers werden dann in **„3. Einkaufsrechnungen“** gegen die Leistungen der Dienstleister verrechnet.

In **„4. Partieabrechnung“** erfolgt dann, nachdem die Dienstleister des Fruchthändlers bezahlt sind, die Bezahlung des Lieferanten in Übersee.

1. Partieerstellung aus Manifest
Für den weiteren Verlauf in dieser Arbeit wird nur der Teilprozess **„Partieerstellung aus Manifest“** betrachtet. Die weiteren Prozesse liefern keinen technischen Mehrwert. Die Schritte entstammen der originalen Testfallspezifikation in Excel.

Nr.	Programm	Arbeitsschritt	Aktion	Eingabe, Prüfwert
1	8.1	Prozess	Einlesen Manifest	Manifestdatei1
2	8.1	Prozess	Einlesen Manifest	Manifestdatei2
3	8.1	Prüfpunkt	Einlesen Manifest	Ergebnis: 84 Paletten / 6.720 Kolli
4	8.1	Prozess	Generieren Partie	Trennen nach Container ID Lager = 100
5	8.2	Prüfpunkt	Partieverwaltung	Ergebnis: 84 Paletten / 6.720 Kolli in 19 Partiepositionen
6	8.2	Prüfpunkt	Partieverwaltung	Detailpaletten müssen da sein
7	8.2	Prüfpunkt	Partieverwaltung	Gutachterkosten = 4,50 x 84 Paletten = 378,00 Euro

Tabelle 5.1: Geschäftsprozess „Partieerstellung aus Manifest“

Die Einträge unter der Spalte **Programm** referenzieren mit der Programmnummer das entsprechende Programm im ERP INTEGRA®.

Die Spalte **Arbeitsschritt** beschreibt, ob es sich um einen „Prozess“ oder „Prüfpunkt“ handelt. „Prozess“ ist hierbei eine abstrakte Beschreibung für einen Prozess im Sinne eines oder mehrerer Arbeitsschritte. Ein „Prüfpunkt“ zeigt z. B. einen Erwartungswert, der später zur Ausführung im Test gegen einen aktuellen Wert geprüft wird.

Die Spalte **Aktion** bedeutet für den Arbeitsschritt → „Prozess“ mit der Aktion → „Einlesen Manifest“: Einlesen einer Manifest-Datei. Die Manifest-Datei ist ein Lieferavis, also die Ankündigung eines Lager- bzw. Warenzugangs.

Die Spalte **Aktion** bedeutet für den Arbeitsschritt → „Prüfpunkt“ mit der Aktion → „Einlesen Manifest“: Überprüfen des Einlesens anhand der Erwartungswerte: 84 Paletten / 6.720 Kolli. Die Erwartungswerte werden gegen aktuelle Werte geprüft. Kolli sind eine Verpackungseinheit im Fruchthandel, sogenannte Verpackungskartons.

Die Spalte **Eingabe, Prüfwert** bedeutet für den „Prozess“ → „Einlesen Manifest“: „Eingabe“ → also das Einlesen einer Datei im CSV-Format.

Die Spalte **Eingabe, Prüfwert** bedeutet für den „Prüfpunkt“ → „Einlesen Manifest“: „Prüfwert“ → also das Prüfen gegen die definierten Sollwerte.

Die Schritte der Nummern 1 bis 3 dienen zum Einlesen der Manifestdateien und Überprüfung des korrekten Einlesens mit den Prüfwerten „84 Paletten/ 6.720 Kolli“.

Durch den Punkt mit der Nr. 4 wird die Partie in INTEGRA® angelegt.

Die letzten Punkte mit den Nummern 5 bis 7 prüfen Sollwerte der Partie-Generierung.

5.5 Vorgehen beim Skripten mit AutoIt

Nachdem der Geschäftsprozess einstudiert wurde, erfolgt das Skripten in AutoIt. Das Skript soll die Tätigkeit des Endanwenders simulieren. Dazu müssen die dabei anfallenden Schritte in die Skriptsprache von AutoIt übertragen werden.

Grundsätzlich ist das Vorgehen dabei:

1. Starten der Anwendung/ Aufrufen einer Unteranwendung aus der bereits gestarteten Anwendung.

2. Warten bis das Fenster der gestarteten Anwendung erschienen ist.

3. Identifizieren eines Steuerelements (Button, Textfeld etc.) auf dem Fenster.
4. Ausführen einer Aktion, z. B. Eingabe eines Wertes oder Prüfen eines Wertes.
5. Wiederholen eine der Vorgängeraktionen.

6 Auswertung

In diesem Abschnitt sollen die Ergebnisse des Pilotprojektes betrachtet werden. Dabei sollen die Vorgaben der ersten Phasen, also die Ziele, Kosten, Risiken und Anforderungen, geprüft werden. Außerdem soll auf die während des Pilotprojektes aufgetretenen weiteren Herausforderungen eingegangen werden. Nach Betrachtung der Herausforderungen erfolgen noch Verbesserungsvorschläge. Insbesondere sollen der Schlüsselwort- und Daten-getriebene Ansatz näher betrachtet werden. Als Beschreibungsgrundlage für die Ansätze werden Code-Ausschnitte aus dem Testskript verwendet.

6.1 Ergebnisauswertung des Pilotprojektes

Das Pilotprojekt wurde beendet. Die bevorstehende Verbreitung von AutoIt steht aber noch aus. Zuvor sind noch Herausforderungen bei der Identifizierung von Steuerelementen der grafischen Oberfläche zu klären.

6.1.1 Bewertung der Ziele

Mit dem Pilotprojekt konnten folgende Ziele erreicht werden:

- Ziel 3 und Ziel 9 konnten erreicht werden. Der zeitliche Rahmen dabei wurde eingehalten.
- Ziel 6 konnte ebenfalls erreicht werden. Der exemplarische Testfall konnte durchgeführt werden.
- Ziel 7 konnte in einem Testbetrieb erfüllt werden. Dazu wurden mehrere AutoIt-Skripte parallel gestartet.
- Ziel 8 konnte erfüllt werden. Zeitmessungen sind in AutoIt nach Stoppuhrprinzip vorhanden.

Folgende Ziele stehen noch aus, da die Verbreitung noch nicht stattgefunden hat:

- Ziel 1 und 2: Eine Verbesserung der Qualität wurde noch nicht gemessen, da das Werkzeug noch nicht im Produktiveinsatz ist.
- Ziel 4: Das Tool ist noch nicht im Produktiveinsatz. Die Robustheit der Testskripte wird sich erst im Produktiveinsatz zeigen können. Es werden also auch noch keine Regressionstests durchgeführt.
- Ziel 5: Die Verbreitung hat noch nicht stattgefunden. Demnach ist das Tool auch noch nicht im Einsatz.

6.1.2 Bewertung der Kosten

Das Maximal-Zeit-Budget von 500 Stunden wurde eingehalten: Der Tester (QS-Consultant) hat für den kompletten Testprozess, bestehend aus den 4 Teilprozessen, 313 Stunden gebraucht. Darin enthalten sind aber auch die Lernphasen für das Testwerkzeug, den Testrahmen und den Geschäftsprozess und dessen Anpassung und Erweiterung in der Testfallspezifikation. Die Zeiten des Vorgenannten wurden nicht separat erfasst. Nach Aussagen des Testers ist für die Erstellung der Testskripte aber ein Verständnis der Testfallspezifikation, bzw. des Geschäftsprozesses, ein wesentlicher Faktor im Zeitbudget.

Die Erstellung des Testrahmens und die Erweiterung hat den Entwickler bzw. den Werkzeugverantwortlichen 40 Stunden gekostet.

Zusammengefasst ist das Pilotprojekt mit 353 Stunden im Zeitrahmen geblieben. Die identifizierten Nacharbeiten wurden nicht in diesem Budget erfasst.

Aussagen über Wartungskosten können also erst nach der Inbetriebnahme getätigt werden.

6.1.3 Bewertung der Anforderungen

Die Anforderung aus Sicht der Firma – Einhaltung des Zeitrahmens – konnte erfüllt werden.

Die Anforderungen der Abteilungen konnten erfüllt werden:

- AutoIt kann von einem QS-Consultant erlernt werden.

- AutoIt arbeitet nicht mit bildbasierter Erkennung.
- Es können Lasttests, also die Simulation von Mehrbenutzerbetrieb, durchgeführt werden.
- Zeitmessungen können und werden derzeit auch in den Log-Dateien durchgeführt.

Die Anforderungen des Testers (QS-Consultant) wurden erfüllt:

- Ein Support durch den Werkzeugverantwortlichen wurde durchgeführt.
- Es wurde eine Anleitung im Wiki bereitgestellt.
- Der Tester musste für das Skripten keine neuen Funktionen anlegen. Er musste nur anwenden. Bei Bedarf wurden neue Funktionen angelegt.

Die Anforderungen des Testers (Entwickler, Projektleiter) wurden ebenfalls erfüllt.

6.1.4 Zusammenfassung

Das Pilotprojekt wurde von den Beteiligten als erfolgreich eingestuft. Das Werkzeug AutoIt wird für die Weiterführung des Vorhabens, nach Klärung der offenen Punkte, eingesetzt. Als nächstes sollen diese offenen Punkte, bzw. Herausforderungen in der Pilotprojektphase, betrachtet werden.

6.2 Herausforderungen in der Pilotprojektphase

In diesem Abschnitt werden die Herausforderungen, die während der Pilotprojektphase aufgetreten sind, betrachtet. Die Herausforderungen bestehen bei folgenden Punkten:

1. Identifizierung von Steuerelementen der grafischen Oberfläche.
2. Schwierigkeiten mit dem Timing.
3. Der Wartungsaufwand beim Codieren der Skripte.
4. Offene Punkte im Testprozess.

6.2.1 Identifizierung von Steuerelementen der grafischen Oberfläche

Die Herausforderung bei der Identifizierung der Steuerelemente beruht auf der Tatsache, dass AutoIt von Haus aus mit „Standard-Windows-Steuerelementen“, die zur Win32Api kompatibel sind, erkennen kann. INTEGRA® wird in seinem Framework, das auf Delphi basiert, entwickelt. Grundlage des darin enthaltenen GUI-Frameworks ist die VCL (Visual Component Library). Im Framework für INTEGRA® sind Steuerelemente durch Objektableitungen teilweise so verändert oder erweitert worden, dass AutoIt diese nicht auf seinem Standardweg ansteuern kann. Workarounds werden notwendig. Beispiele für Steuerelemente, die nicht identifiziert werden können, sind Tabbed Notebooks (Panels mit Registerkarten) und insbesondere Grid-Komponenten. Workarounds sind hier dann die Navigation zu diesen mit der TAB-Taste, Pfeil-Tasten oder die Benutzung von Tastatur-Kombinationen bzw. Hotkeys. Insbesondere der Workaround für die Interaktion mit Grid-Komponenten ist momentan sehr aufwendig gestaltet und besteht aus den Schritten:

- Den Inhalt des Grids mit einer Standard-Funktion von INTEGRA® in eine Excel-Datei exportieren. Dieser Vorgang wird natürlich mit den Funktionen zur Interaktion mit der Oberfläche von AutoIt durchgeführt.
- Die Excel-Datei mit Win-Api-Excel-Funktionen öffnen und in eine CSV-Datei speichern.
- Die CSV-Datei mit File-I/O in eine AutoIt-Variable einlesen.
- Den Inhalt der Variablen in ein zweidimensionales Array parsen.
- Dann kann mit zeilen- und spaltenweisen Zugriffsfunktionen auf das Array zugegriffen werden.

Diese Umsetzung funktioniert, ist aber anfällig für Auswirkungen von außen durch Effekte der Systemebene. Außerdem dauert die Ausführung sehr lange.

Weitere Schwierigkeiten bestehen bei der Auffindung von Steuerelementen, bspw. wenn keine eindeutigen Identifikatoren vergeben wurden. Als Beispiel sei hier ein Formular gegeben mit zehn zur Laufzeit generierten Textfeldern der gleichen Art. Als einziges Unterscheidungsmerkmal wird eine interne ID vergeben. Allerdings ändert sich die Vergabe der Reihenfolge von Programmausführung zu Programmausführung. Beim Ausführen der Testakte wäre die Reihenfolge also nicht sichergestellt. Als Workaround wurde hier teilweise eine Navigation

mit Tab und Pfeiltasten versucht. Ändert sich allerdings die Tab-Reihenfolge oder die Anzahl Tastenanschläge für die Pfeiltaste, kommt es zum Abbruch.

Lösungsversuch
Aufgrund der Herausforderungen, insbesondere der sich ändernden dynamischen Identifikationsmerkmale, wurde eine Lösung zur Identifizierung der Steuerelemente analysiert. Es wurde dafür auch eine Lösung gefunden. Diese ist in der finalen Testphase.

Bei dieser Lösung macht sich die Firma PASCAL den Zugriff auf den Quelltext von INTEGRA® zu Nutzen. Für die Identifizierung der Steuerelemente wurde bzw. wird eine eigene Lösung geschaffen. Dafür wurde Funktionalität in INTEGRA® eingebaut. Sinngemäß werden bei dieser Lösung die Steuerelemente durch INTEGRA® preisgegeben. Mit einer Tastenkombination wird dazu eine Liste mit eindeutigen Namen von den Steuerelementen ausgegeben, deren Anwendungsfenster von INTEGRA® gerade aktiv auf dem Desktop ist. Die Liste wird in die Zwischenablage gespeichert. Aus der Liste muss das gewünschte Steuerelement gesucht und ausgewählt werden. Der eindeutige Name des Steuerelements dient als Eingabe für eine nächste Funktion in INTEGRA®. Dazu wird der Name mit einem Steuerbefehl in die Zwischenablage kopiert. INTEGRA® liefert dann den für den aktuellen Programmlauf gültigen Handle des Steuerelements. Mit diesem Handle können die Funktionen zum Umgang mit den Steuerungselementen umgehen.

Mit diesem Vorgehen lässt sich nun ein Großteil der Steuerelemente bedienen. Eine Beeinflussung des Testablaufes durch diese Kommunikation ist höchst unwahrscheinlich (< 0,001%).

6.2.2 Schwierigkeiten mit dem Timing

Während der Pilotphase ergaben sich Herausforderungen beim Timing. In den ersten Versuchen der Umsetzung wurde viel Gebrauch von festen Delays gemacht. Dieses Vorgehen führte allerdings zu häufigen Abbrüchen bei Zeitüberschreitungen. Als Lösung wurden die Funktionen zur Interaktion mit der GUI mit Timing-Funktionalität erweitert. Dabei läuft ein Timeout ab. Das Timeout ist auf einen Standardwert gesetzt, kann aber per Parameter geändert werden. Intern prüft die Funktion nach dem Polling-Prinzip, ob das gewünschte Objekt (Fenster oder Steuerelement) verfügbar ist. Als Nebeneffekt werden Zeitmessungen damit exakter.

Ein weiteres Problem mit dem Timing ergab sich beim Öffnen und Warten auf Fenster. Häufig kam es dabei zu Abbrüchen. In solchen Fällen meldete das Fenster schon zurück, dass es existiert, obwohl das Fenster noch nicht gezeichnet wurde. Das Fenster wurde von den Funk-

tionen gefunden, die Aktionen (Eingaben, Tastendrücke) gingen aber ins Leere und sorgten schlussendlich für den Abbruch der Ausführung des Skriptes. Zur Lösung hierfür wurde eine UDF (User Defined Function) von AutoIt verwendet, die eine Windows-Message an ein Fenster/ Anwendung schicken kann. Die Windows-Message ist an ein Timeout gebunden. Im konkreten Fall wurde als Windows-Message jene gewählt, die das gewünschte Fenster der Anwendung dazu veranlasst, sich neuzuzeichnen. Diese Funktion wird vom Betriebssystem ständig an das Programm gesendet, wenn es zu grafischen Veränderungen durch z. B. Überlagerung kommt. Auch hier kann eine Beeinflussung des Testprozesses ausgeschlossen werden. Das Problem konnte mit dieser Funktion gelöst werden.

6.2.3 Offene Punkte im Testprozess und weiteres

In diesem Abschnitt werden Punkte aufgezählt, die für die Zukunft geprüft werden sollten. Generell basieren diese Vorschläge aus Vorgaben der Basisliteratur [LS12], [WSLR14] und werden im Zusammenhang als wichtige, kritische Punkte erwähnt.

Testumgebung
Bisher wird in der Umgebung getestet, in der auch Entwickler und Kunden Zugriff haben. Dadurch kommt es zu Problemen, bspw. durch geänderte Konfigurationseinstellungen, die parallel von einem Kollegen gesetzt wurden.

Verfahren für detaillierte Testfallspezifikation
Verfahren für detaillierte Testfallspezifikation, wie Äquivalenzklassenbildung oder Entscheidungstabellentechnik, um Testeingaben und Testfälle systematisch zu ermitteln, wurden bisher nicht angewendet. Bisher wird die Kernfunktionalität abgedeckt. Bei den umfangreichen geschäftsprozessbasierten Testfällen werden solche, wie die vorher erwähnten Verfahren, als zu aufwendig angesehen.

Das Verfahren des anwendungsfallbasierten Testens wurde für die Zukunft nicht ausgeschlossen. Für das Pilotprojekt wurde aus Einfachheitsgründen auf geschäftsprozessbasiertes Testen mit der Spezifikation von Testfällen in einem Tabellenkalkulationsprogramm gesetzt. Die einzelnen Tabelleneinträge aus Tabelle 5.1 könnten aber auch für entsprechende Use Cases in einem Use-Case-Diagramm stehen.

Bezug von der Testfallspezifikation zum Testskript ist nicht eindeutig
Die Testfallspezifikation ist nicht eindeutig auf das zugehörige Testskript abgebildet. Das erschwert das Verständnis und erfordert Nacharbeit bei etwaigen Wartungsarbeiten etc. Eine

Idee für die Erweiterung des Testrahmens wäre hier zum einen der Schlüsselwort-getriebene Ansatz bei der Codierung der Testskripte. Des Weiteren könnte man versuchen, ähnlich wie es die Werkzeuge [Tho18] und [Fou18] umsetzen, Testfallspezifikation und Testskript in einem durchzuführen.

Bezug während der Ausführung des Testskriptes
Eine Funktion zur Anzeige des nächsten Testschritts in Form eines Tool-Tipps wurde eingebaut. Allerdings ist auch hier der Bezug nicht eindeutig zu erkennen. Der Ausgabetest des Tool-Tipps stimmt nicht überein und das Timing, also der Zeitpunkt und die Anzeigedauer, sind nicht korrekt.

Auffinden der Wirkung, die den Abbruch des Skriptes verursacht
Das Auffinden der Wirkung, die den Abbruch des Skriptes verursacht, ist aufwendig. Sofern man den Testfall und das Testskript kennt, ist das ein kleineres Problem. Ist dieses Wissen nicht vorhanden, muss mit Informationen aus der Log-Datei und dem Suchen im Testskript die entsprechende Stelle des Ausstiegs erst gesucht werden. Dann können weitere Ursachen recherchiert werden. Ein gewisser Aufwand, der bei anderen Werkzeugen, wie TestComplete [Tes18], nicht besteht. Bei TestComplete reicht ein Klick auf den Fehlereintrag im Report, um direkt zur Stelle des Ausstiegs zu springen. Für die Zukunft ist eine ähnlich komfortable Umsetzung bei der Integration von AutoIt in INTEGRA® geplant.

6.2.4 Der Wartungsaufwand beim Codieren der Skripte

Der Wartungsaufwand fällt mit Blick auf die zuvor beschriebenen Herausforderungen geschätzt entsprechend hoch aus. Die Teststufe (Systemebene) macht es hier nicht einfacher durch die hier möglichen äußeren Einflüsse. Es können nicht alle Ereignisse, die eintreten und zum Abbrechen der Ausführung von Skripten führen, bedacht werden. Es kann versucht werden, die Testakten möglichst robust zu gestalten. Natürlich unter Berücksichtigung der Kosten, um den Wartungsaufwand gering zu halten.

Als Empfehlung sollte außerdem der Weg der Identifizierung von Steuerelementen durch INTEGRA® konsequent genutzt werden. Die Funktionalität der Identifizierung durch INTEGRA® sollte auf bisher nicht unterstützte Steuerelemente erweitert werden, insbesondere für Grid-Komponenten.

Als weitere Möglichkeit sollte die Entwicklung der Testskripte nach dem Schlüsselwort- und Daten-getriebenen Ansatz geprüft werden [SBGB15]. Mit dem konsequenten Einsatz dieser

beiden Methoden lassen sich die Wartungsaufwände reduzieren. Beispiele beider Ansätze erfolgen im Abschnitt 6.3.

6.3 Verbesserungsvorschläge

In diesem Abschnitt erfolgt eine Aufführung von Verbesserungsvorschlägen zur Programmierung der Testskripte. Dabei werden der Schlüsselwort- und der Daten-getriebenen Ansatz aus [SBGB15] betrachtet.

6.3.1 Schlüsselwort-getriebener Ansatz

Beim Schlüsselwort-getriebenen Ansatz werden logisch zusammenhängende Teile gekapselt. Mit der Möglichkeit zur Erstellung von Funktionen kann mit AutoIt dieser Ansatz sinngemäß umgesetzt werden. Als Funktionsname sollte dann ein sinnvoller Name gewählt werden, der in den weiteren Testmitteln – Testfallspezifikation, Anforderungsdokumente usw. – ebenso verwendet wird.

Als Beispiel soll hier ein Ausschnitt aus dem in 5.4 beschrieben Geschäftsprozess bzw. Testfaltspezifikation dienen:

Nr.	Programm	Arbeitsschritt	Aktion	Eingabe, Prüfwert
1	8.1	Prozess	Einlesen Manifest	Manifestdatei1

Tabelle 6.1: Ausschnitt aus dem Geschäftsprozess „Partieerstellung aus Manifest“

Der entsprechende Teil der Testakte sieht folgendermaßen aus: (Kommentare werden in AutoIt mit einem ; eingeleitet.)

```
1
2   $Import_File = IniRead("Import_File_1")
3
4
5   If ($Import_File <> "") Then
6
7
8     ; Button "Manifest einlesen" drücken
9     ControlFocus("TFormReisen", "", "BtnEinlesen")
10    Send("{SPACE}")
11
12    ; Eingabe des Pfads zu Datei 1
13    TextPaste($Import_File)
14
15
16    ; Button "Öffnen" drücken
```

```
    ControlFocus("TFormÖffnen", "", "BtnÖffnen")
    Send("{SPACE}")

    ; Fenster "Import Manifest" schließen
    ControlFocus("TFormImportManifest", "", "BtnClose")
    Send("{SPACE}")

  EndIf
.
```

Listing 6.1: Quelltext des Prozessschrittes „Importiere Datei 1“

Die Anwendung des Schlüsselwort-getriebenen Ansatzes führt zu folgendem Code:

```

  Func ImportiereManifestDatei($PfadManifestDatei)

    If ($PfadManifestDatei <> "") Then

      ButtonManifestEinlesen()

      EingabePfad($PfadManifestDatei)

      ButtonOeffnen()

      FensterImportManifestSchliessen()
    EndIf

  EndFunc
.
```

Listing 6.2: Definition der Funktion „ImportiereManifestDatei($PfadManifestDatei)“

Der Code wird übersichtlicher, ist wiederverwendbar und parametrisierbar. Außerdem konzentriert sich, für den Fall eines Fehlers im Code, die Wartungsarbeit auf diese Funktion. Der Code-Ausschnitt reduziert sich auf den Aufruf:

```

  ImportiereManifestDatei($PfadManifestDatei)
.
```

Listing 6.3: Aufruf der Funktion „ImportiereManifestDatei($PfadManifestDatei)“

6.3.2 Daten-getriebener Ansatz

Beim Daten-getriebenen Ansatz werden Testeingaben und vorausgesagte Ergebnisse in einer Tabelle gespeichert. Mit Zugriffsfunktionen wird auf die Zeilen der Tabelle zugegriffen. Generell stellt eine Datenzeile der Tabelle einen Testfall dar.

```
1
2   ; Editiere Reise: voraussichtliche Ankunftszeit setzen
3   ControlFocus("FormReiseStamm"), "", "TBDBDateTimeEditDom1")
4   Send("21.01.2017")
5   Send("{ENTER}")
6
7   ; Einlagerung setzen
8   ControlFocus("TFormReiseStamm", "", "TBDBSearchEditDom2")
9   Send("DORT")
10  Send("{ENTER}")
11
12  ; Datensatz speichern
13  IntegraSave()
14
15 .
```

Listing 6.4: Quelltext des Prozessschrittes „Generieren Partie“

Anwendung des Schlüsselwort-getriebener Ansatzes:

```
1
2
3   Func GenerierenPartie($Datum, $Lager)
4
5     SetzeVoraussichtlicheAnkunftszeit($Datum)
6
7     SetzeEinlagerung($Lager)
8
9     IntegraSave()
10
11  EndFunc
12
13 .
```

Listing 6.5: Definition der Funktion „GenerierenPartie($Datum, $Lager)“

Anwendung des Daten-getriebenen Ansatzes mit der Datentabelle und den Funktionen für den Zugriff auf die jeweilige Datenspalte:

Datum	Lager
21.01.2017	DORT
10.02.2017	HAMB
15.03.2017	BIEL

```
1
2
3   GenerierenPartie(GetDatum(), GetLager())
4
5 .
```

Listing 6.6: Aufruf der Funktion „GenerierenPartie(GetDatum(), GetLager())“

Mit der Anwendung des Daten-getriebenen Ansatzes kann nun mit der Datentabelle auf übersichtlichem Wege das Testskript gesteuert werden. Eine Trennung von Daten und Logik wird erreicht. Die beiden Ansätze – Daten- und Schlüsselwort-getriebener Ansatz – lassen sich gut kombinieren.

Eine Trennung von Daten und Logik ist zuvor schon durch den Einsatz einer Ini-Datei erfolgt. Allerdings haben Ini-Dateien die Eigenschaft unübersichtlich zu werden, sofern diese mit der Zeit anwachsen. Mit der Zufuhr der Daten und Steuerung des Skripts durch eine Datei im tabellarischem Format, hat man eine aufgeräumtere Lösung. Eine Zeile der Datentabelle kann hier Daten für einen Durchlauf des Skripts enthalten. Denkbar sind auch Daten, die zur Steuerung des Skripts dienen.

7 Fazit

In diesem Kapitel erfolgt eine Zusammenfassung der Ergebnisse und ein Ausblick für die zukünftigen Aktivitäten.

7.1 Zusammenfassung

Die technische Machbarkeit des Vorhabens unter Einhaltung des zeitlichen Rahmens, konnte mit AutoIt erreicht werden. Großen Anteil daran trägt die Möglichkeit, durch INTEGRA® die Identifizierung der Steuerungselemente durchzuführen.

Ein weiterer nicht zu vernachlässigender Punkt ist das Wissen der Geschäftsprozesse. Diese werden bei ERP-Systemen schnell sehr komplex. Eine detaillierte Dokumentation und das Wissen, wie diese im ERP-System umzusetzen ist, ist von sehr hoher Wichtigkeit. Denn generell sind die Geschäftsprozesse so sehr integriert, dass sich inhaltliche Fehler nicht ohne weiteres rückgängig machen lassen. Was zur Folge hat, dass der Geschäftsprozess erneut von Beginn an durchgeführt werden muss.

Sehr wichtig ist dafür das Vorhandensein von Anforderungen, um Testfälle überhaupt spezifizieren zu können. Sind die Anforderungen nicht gesammelt oder verteilt auf Dokumente und in den Köpfen von Kunden und Kollegen, müssen diese Informationen erst eingesammelt und aufwendig in zusammenhängende Form gebracht werden.

Aber gerade bei der Automatisierung sind korrekte Testfallspezifikationen wichtig. Falsche unvollständige Anforderungen führen nicht zu einem korrekten Geschäftsprozess und machen somit die Automatisierung unmöglich [SBGB15].

Die Art von geschäftsprozessbasierten Tests und die Komplexität der Geschäftsprozesse bei ERP rücken den Teil der Testfallspezifikation in den Vordergrund. Das Werkzeug wird fast nebensächlich. Die Techniken bei den meisten dieser Tools sind Schlüsselwort-getrieben und Daten-getrieben [SBGB15] (vgl. hierzu auch das Robot Framework [Fou18] und Gauge [Tho18]).

Reines Capture & Replay und Skripten von oben nach unten wurde längst als ineffizient identifiziert [SBGB15].

Auf der Ebene des Systemtests kommt ein weiterer Punkt zum Tragen, der sich unmittelbar auf die in der Vergangenheit gewählte Teststrategie zurückführen lässt. Der reaktive Testansatz mag anfangs seine Vorzüge haben, weil der Overhead an Spezifikationsarbeiten entfällt, doch sind die Aufwände beim nachgelagerten Testen höher. Wer sich vorher Gedanken macht, wie man hinterher die Software am einfachsten testen kann, entwickelt die Software gleich so, dass sie testbar wird [LS12].

Dennoch konnte gezeigt werden, dass sich mit AutoIt ein komplexer Geschäftsprozess automatisieren lässt. Die eigens entwickelte Lösung zur Identifizierung der Steuerungselemente in Kombination mit AutoIt ist dabei, neben dem Vorhandensein von ausreichender Dokumentation und dem Wissen der geschäftsprozessbasierten Testfälle, von essentieller Bedeutung.

7.2 Ausblick

Für die Zukunft sollten die noch nicht erreichten Zielsetzungen und Herausforderungen aus der 6 Auswertung geklärt werden. Insbesondere die Verbesserung der Qualität unter Berücksichtigung der Kostenaspekte muss nach der Einführung des Werkzeuges geprüft bzw. eingehalten werden.

Die Eigenentwicklung in Kombination mit AutoIt zur Identifizierung der Steuerungselemente sollte konsequent genutzt und ausgebaut werden.

Die in 6.3 Verbesserungsvorschläge beschriebenen Techniken sollten eingeführt werden, um die Wartung gering zu halten. Außerdem wird mit dem Daten-getriebenen Ansatz eine gute Möglichkeit zur Variation von Testfällen über die Testeingaben erreicht.

Für die Zukunft plant die Firma PASCAL die Integration der Testmittel in das ERP INTEGRA®.

Nach Klärung der Nacharbeiten steht die Verbreitung des Werkzeuges an. Das Ausrollen soll nach folgender Strategie erfolgen: Pro Abteilung wird ein QS-Verantwortlicher benannt, der Hauptverantwortlicher und Ansprechpartner für die Kollegen ist. Diese Werkzeugverantwortlichen sollen in ihrer Abteilung die Verteilung und die Haupttätigkeit mit dem Werkzeug AutoIt übernehmen. In der Verbreitungsphase und darüber hinaus stehen die Mitarbeiter aus dem Pilotprojekt als Support zur Verfügung.

Für die Zukunft wurde bereits eine Rubrik für das Testen mit AutoIt im Firmen-Wiki eingerichtet. Erste Schnellstartanleitungen und Best Practices sind bisher in die Rubrik eingeflossen. Für die Zukunft sind regelmäßige Meetings – alle zwei Monate – für den Wissensaustausch eingeplant.

A Anhang

INHALT

A.1 Evaluation Werkzeuge

			TestComplete			Eigen (AutoIT)			JitBit MacroRecorder		
Gruppe	**Kriterium**	**Gewichtung**	**Punkte**	**Gesamtwert**	**Summe**	**Punkte**	**Gesamtwert**	**Summe**	**Punkte**	**Gesamtwert**	**Summe**
Testautomat	Bedienung	3	8	24		6	18		7	21	
	Installation	4	6	24		9	36		7	28	
	Wartbarkeit	5	8	40		8	40		7	35	
	Einbindbar in INTEGRA-Framework	4	2	8		10	40		6	24	
	Testautomation auf Anwendungslogik	5	3	15		5	25		1	5	
	SinglePerformance-Messung	3	0	0		10	30		0	0	
	rudimentäre Lasttests	1	0	0		10	10		0	0	
	Funktioniert Abseits von INTEGRA	1	10	10		1	1		4	4	
	Summe				**121**			**200**			**117**
Testautomat-Framework	Bedienung	3	10	30		8	24		1	3	
	Wartbarkeit	3	7	21		9	27		1	3	
	Einbindbar in INTEGRA-Framework	5	1	5		10	50		1	5	
	Installation	1	6	6		9	9		1	1	
	Auswertungen	3	10	30		5	15		1	3	
	Aufwand der Erstellung	5	10	50		1	5		1	5	
	Summe				**142**			**130**			**20**
Mitarbeiter	KnowHow - vorhanden	5	2	10		10	50		4	20	
	Schulungsaufwand	4	7	28		4	16		4	16	
	Summe				**38**			**66**			**36**
Wirtschaftlichkeit	Lizenzkosten	2	3	6		10	20		7	14	
	Laufende Kosten	4	3	12		10	40		7	28	
	Zeitaufwand Einarbeitung	2	8	16		8	16		7	14	
	Schulungsaufwand	2	7	14		7	14		7	14	
	Gesamtzeitaufwand Erstellung	4	5	20		5	20		2	8	
	Ausfallrisiko Drittanbieter	1	3	3		8	8		5	5	
	Projektlaufzeit - Prototyp Testautomat	2	8	16		6	12		7	14	
	Summe				**87**			**130**			**97**
Grundbedingung	Windows		X			X			X		
	Kompatibel zu INTEGRA		X			X			X		
SUMME					**388**			**526**			**270**

Abbildung A.1: Nutzwertanalyse zum Projekt Testautomation – 10.10.2017

A.2 Au3Info – AutoIt-Spy-Tool

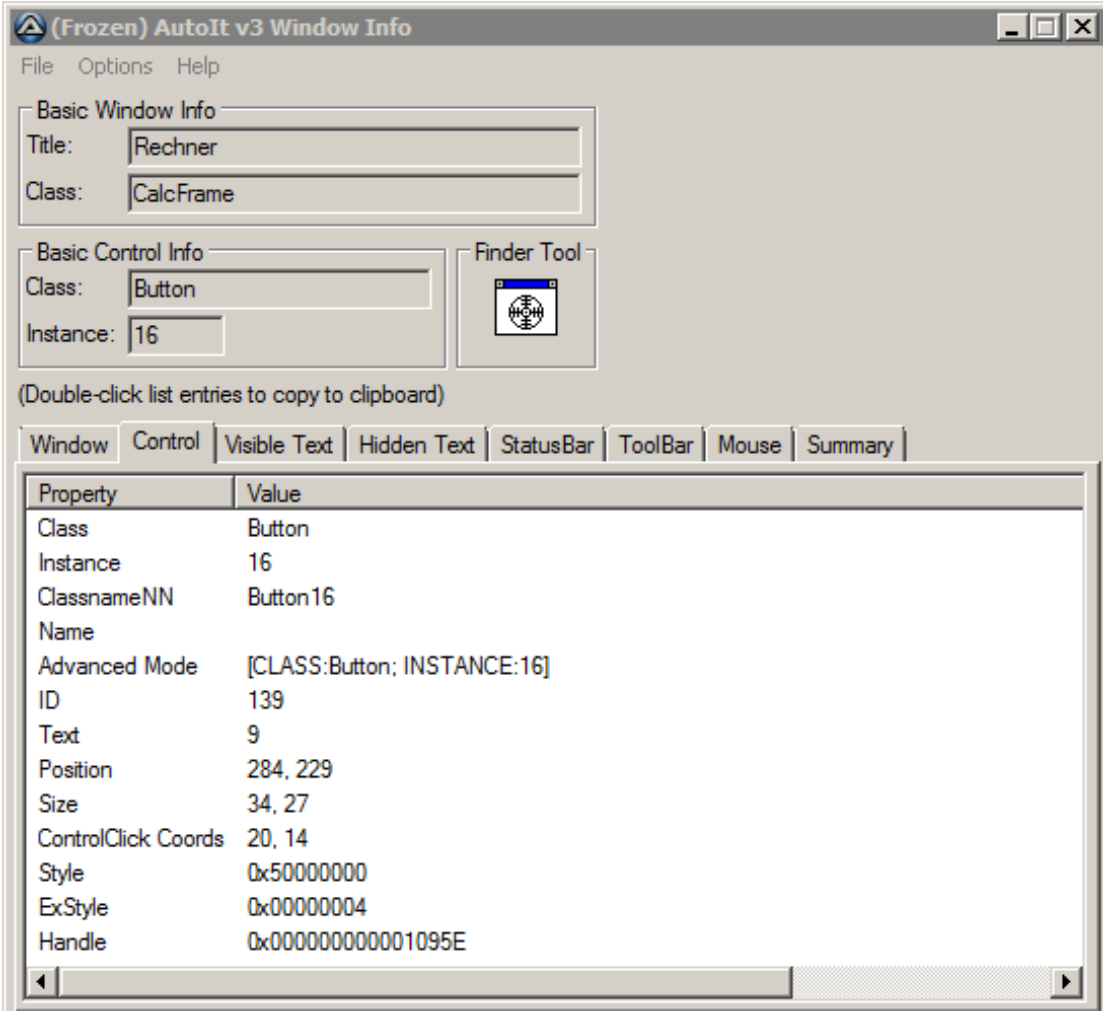

Abbildung A.2: AutoIt-Spy (Au3Info) [Aut18]

A.3 Editor – SciTE4AutoIt3

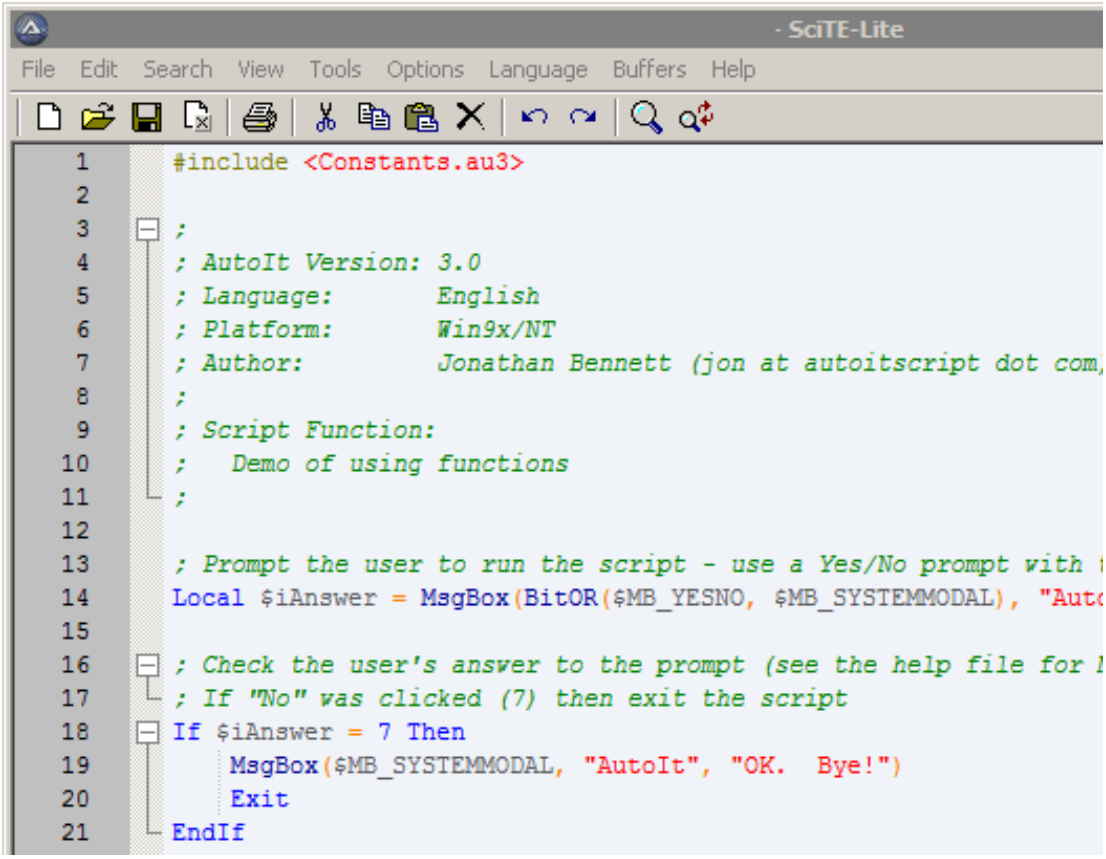

Abbildung A.3: SciTE4AutoIt3 [Aut18]

A.4 AutoIt Skript Compiler Aut2Exe

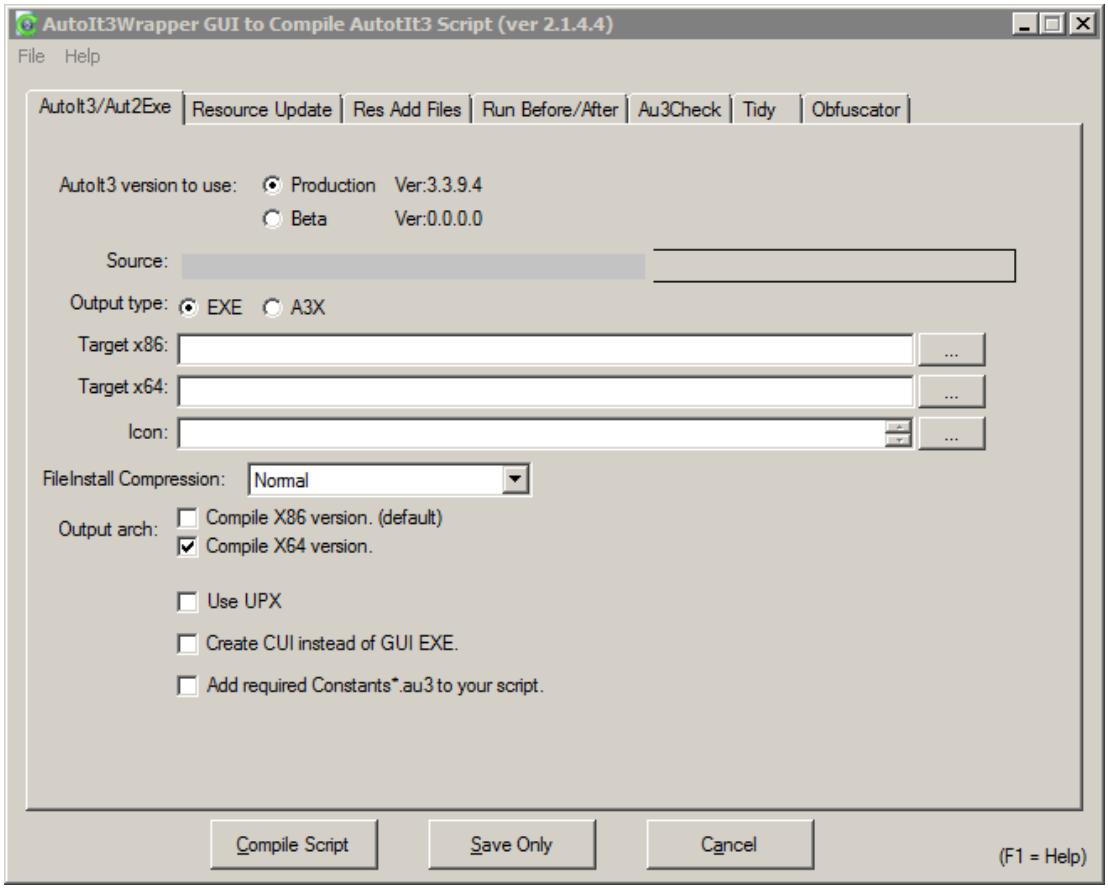

Abbildung A.4: Aut2Exe Dialog zur Skriptkompilierung [Aut18]

Literaturverzeichnis

[AM17] Abts, Dietmar ; Mülder, Wilhelm: *Grundkurs Wirtschaftsinformatik - Eine kompakte und praxisorientierte Einführung*. 9. Berlin Heidelberg New York : Springer-Verlag, 2017. – ISBN 978–3–658–16379–2

[Aut18] AutoIt: *AutoIt Scripting Language*. https://www.autoitscript.com/site/autoit/. Version: 2018. – [Online; abgerufen 11-Mai-2018]

[Bay13] Bayer, Martin: *Modernes ERP - eine Frage der Architektur*. https://www.computerwoche.de/a/modernes-erp-eine-frage-der-architektur,2504748,3. Version: 2013. – [Online; abgerufen 05-Juni-2018]

[BF12] Borjesson, Emil ; Feldt, Robert: Automated system testing using visual gui testing tools: A comparative study in industry. In: *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* IEEE, 2012, S. 350–359

[DFFS13] Duan, Jiaqi ; Faker, Parwiz ; Fesak, Alexander ; Stuart, Tim: Benefits and drawbacks of cloud-based versus traditional ERP systems. In: *Proceedings of the 2012-13 course on Advanced Resource Planning* (2013)

[Fou18] Foundation, Robot F.: *Robot Framework*. http://robotframework.org/. Version: 2018. – [Online; abgerufen 27-Mai-2018]

[Fow11] Fowler, Martin: *Eradicating Non-Determinism in Tests*. https://martinfowler.com/articles/nonDeterminism.html#LackOfIsolation. Version: 2011. – [Online; abgerufen 28-Mai-2018]

[Fow13] Fowler, Martin: *BroadStackTest*. https://martinfowler.com/bliki/BroadStackTest.html. Version: 2013. – [Online; abgerufen 4-Juni-2018]

[GTBe17] German Testing Board e.V., Arbeitsgruppe G.: ISTQB/GTB Standardglossar der Testbegriffe. In: *ISTQB* (2017). http://www.

german-testing-board.info/wp-content/uploads/2016/06/ISTQB%C2%AEGTB-Standardglossar-der-Testbegriffe-Deutsch-Englisch.pdf

[Kee15] Kees, Alexandra: *Open Source Enterprise Software - Grundlagen, Praxistauglichkeit und Marktübersicht quelloffener ERP-Systeme.* Berlin Heidelberg New York : Springer-Verlag, 2015. – ISBN 978–3–658–09805–6

[KRG00] Klaus, Helmut ; Rosemann, Michael ; Gable, Guy G.: What is ERP? In: *Information systems frontiers* 2 (2000), Nr. 2, S. 141–162

[LLS15] Laudon, Kenneth C. ; Laudon, Jane P. ; Schoder, Detlef: *Wirtschaftsinformatik - Eine Einführung.* München : Pearson Studium, 2015. – ISBN 978–3–868–94269–9

[LS12] Linz, Tilo ; Spillner, Andreas: *Basiswissen Softwaretest - Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard.* Heidelberg : dpunkt.verlag, 2012. – ISBN 978–3–864–91202–3

[Lüb18] Lübke, Daniel: *Testen von ausführbaren Geschäftsprozessen.* https://www.innoq.com/de/articles/2012/10/auf-nummer-sicher/. Version: 2018. – [Online; abgerufen 18-Mai-2018]

[NTD12] Nazemi, Eslam ; Tarokh, Mohammad J. ; Djavanshir, G R.: ERP: a literature survey. In: *The International Journal of Advanced Manufacturing Technology* 61 (2012), Nr. 9-12, S. 999–1018

[San14] Santos, Pablo: *Don't Develop GUI Tests, Teach Your App To Test Itself!* http://www.drdobbs.com/testing/dont-develop-gui-tests-teach-your-app-to/240168468. Version: 2014. – [Online; abgerufen 15-Mai-2018]

[SBGB15] Seidl, Richard ; Bucsics, Thomas ; Gwihs, Stefan ; Baumgartner, Manfred: *Basiswissen Testautomatisierung - Konzepte, Methoden und Techniken.* 2. Heidelberg : dpunkt.verlag, 2015. – ISBN 978–3–864–91706–6

[SBS12] Sneed, H.M. ; Baumgartner, M. ; Seidl, R.: *Der Systemtest: von den Anforderungen zum Qualitätsnachweis.* Hanser, 2012 https://books.google.de/books?id=rRpUtQAACAAJ. – ISBN 9783446426924

[Sci18] Scintilla: *SciTE – A free source code editor for Win32 and X.* https://www.scintilla.org/SciTE.html. Version: 2018. – [Online; abgerufen 22-Mai-2018]

[Sof18] Software, Jitbit: *JitBit Macro Recorder*. https://www.jitbit.com/macro-recorder/. Version: 2018. – [Online; abgerufen 06-Juni-2018]

[Sub18] Subverison, Apache: *Enterprise-class centralized version control for the masses*. https://subversion.apache.org/. Version: 2018. – [Online; abgerufen 25-Mai-2018]

[Tes18] TestComplete, SMARTBEAT: *The Easiest-to-Use Automated UI Testing Tool*. https://smartbear.com/product/testcomplete/overview/. Version: 2018. – [Online; abgerufen 25-Mai-2018]

[Tho18] ThoughtWorks: *Gauge Test Automation*. https://gauge.org/. Version: 2018. – [Online; abgerufen 27-Mai-2018]

[WSLR14] Winter, Mario ; Spillner, Andreas ; Linz, Tilo ; Rossner, Thomas: *Praxiswissen Softwaretest - Testmanagement - Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard*. 4. Heidelberg : dpunkt.verlag, 2014. – ISBN 978–3–864–91516–1

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 7. Juni 2018 Torben Kuhlmann