FERDINAND PORSCHE
FERN FH

Testautomatisierung des ERP-Systems bei Wewalka Frischteig GmbH

Bachelorarbeit

eingereicht von: **Claus Koizar**
Matrikelnummer: 52006076

im Fachhochschul-Bachelorstudiengang Wirtschaftsinformatik (0470)
der Ferdinand Porsche FernFH

zur Erlangung des akademischen Grades eines
Bachelor of Arts in Business

Betreuung und Beurteilung: Dipl.-Ing Dr. Werner Toplak

Wiener Neustadt, Mai 2023

Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.
2. dass ich diese Bachelorarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe.

Leobersdorf, 13.05.2023

Signiert von:	CLaus Koizar
Datum:	13.05.2023 09:44:22

A TRUST

Dieses Dokument ist digital signiert!

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Die Kapitel 2 bis 3 der vorliegenden Bachelorarbeit wurden im Rahmen der Lehrveranstaltung „Bachelor Seminar 1“ eingereicht und am 02.02.2023 als Bachelorarbeit 1 angenommen.

Kurzzusammenfassung: Testautomatisierung des ERP-Systems bei Wewalka Frischteig GmbH

Die Firma Wewalka GmbH arbeitet mit dem ERP-System Dynamics 365 for Finance & Operations von Microsoft. Bisher wurden Testfälle immer von Mitarbeiter*innen aus den Fachabteilungen durchgeführt. Das Ziel ist die Einführung eines automatisierten Testsystems, das bei Updates und Implementierung neuer Funktionen das System auf Korrektheit überprüft. Hier stellte sich die Frage, ob durch den Einsatz der Testautomatisation der zeitliche Aufwand um den Faktor 5 verringert werden kann.

Die Testautomatisierung wurde durch drei Testfälle mittels Aufgabenaufzeichnung und anschließendem Testwerkzeug von Microsoft umgesetzt, begleitet wurde diese durch drei Testfälle, die mittels Modultests abgebildet wurden.

Durch Zeitaufzeichnung der automatisierten Tests und der manuellen Durchgänge konnten die Ansätze verglichen werden und zeigten, dass der zeitliche Faktor nicht um den Faktor 5 verringert werden konnte. Jedoch konnten andere positive Aspekte den Einsatz der Testautomatisierung rechtfertigen, wie beispielsweise kein Ressourcenverbrauch in Fachabteilungen oder eine zentrale Speicherung der durchgeführten Testfälle.

Schlagwörter:

Testautomatisierung, Regression Suite Automation Tool, Microsoft Dynamics 365 for Finance & Operations, Modultests, Automatisierung, Testdokumentation

Abstract: Test Automation of the ERP-System for Wewalka Frischteig GmbH

Microsoft Dynamics 365 for Finance & Operations is the ERP-System used at Wewalka GmbH. Until now tests were performed by employees. The main goal was to setup an automated test system for platform updates and release of new implemented functions. The following thesis should be answered with the implementation of the automated test system: “Can we reduce the time effort of software tests by implementing a test automation by factor 5?”

The test automation was implemented with three test cases by “Record and playback” and three test cases, transformed into unit tests.

By time recording of the automated tests and manual tests these two approaches were compared. It showed, that the time effort is not reduced by factor 5 when using test automation, but other positive aspects were found during the use of the test automation, such as reducing resource consumption or a positive effect of centralised storage of former test cases.

Keywords:

Test Automation, Regression Suite Automation Tool, Microsoft Dynamics 365 for Finance & Operations, Unit Tests, Automation, Test Documentation

Inhaltsverzeichnis

1. EINLEITUNG **1**

1.1 Ausgangssituation **1**

1.2 Zielsetzung und Abgrenzung **2**

1.3 Persönliches Interesse **3**

1.4 Methodik und Aufbau **3**

1.5 Struktur der Arbeit **4**

2. GRUNDLAGEN UND DERZEITIGER STAND VON WISSENSCHAFT UND TECHNIK **5**

2.1 Kapitelübersicht **5**

2.2 Hintergrund automatisierter Tests **5**

2.2.1 Funktionale Tests 7

2.2.2 Smoke Test 7

2.2.3 Unit Test 7

2.2.4 Sanity Test 7

2.2.5 Integration Test 7

2.2.6 Regression Test 7

2.2.7 Nicht-funktionale Tests 8

2.2.8 Performance Test 8

2.2.9 Security Test 8

2.2.10 Usability Test 8

2.2.11 System Test 9

2.2.12 User Acceptance Test 9

2.3 Continuous Integration (CI) **10**

2.4 Aktueller Stand der Technik **12**

2.4.1 MSTestV2 12

2.4.2 NUnit 13

2.4.3 xUnit.Net 13

2.4.4 Entscheidung Modultest-Umgebung 14

2.4.5 Selenium 14

2.4.6 Appium 15
2.4.7 Protractor 15
2.4.8 SikuliX 15
2.4.9 Tricentis Tosca 15
2.4.10 Regression Suite Automation Tool (RSAT) 16
2.4.11 Entscheidung Integrationstest-Umgebung 16
2.5 Aktueller Stand der Wissenschaft 17
2.5.1 Agile Software-Entwicklung 17
2.5.2 Test Driven Development (TDD) 19
2.5.3 Entscheidungsgrundlagen zur Testautomatisierung 20
2.5.4 Hindernisse bei der Testautomatisierung 22
2.5.5 Best Practices / Bewährte Vorgehensweisen 24
2.5.6 Testautomatisierung mit Artificial Intelligence/Machine Learning 26
2.5.6.1 AccelQ 27
2.5.6.2 Data Dog 27
2.5.6.3 Katalon Studios 27
2.5.6.4 Mabl 27
2.5.6.5 Perfecto Scriptless 27

3. KONZEPTIONELLER VORGEHENS- UND LÖSUNGSANSATZ 28
3.1 Kapitelübersicht 28
3.2 Methodenübersicht 28
3.2.1 Primärdatenerhebung 28
3.2.2 Proof-of-Concept 30

4. ANWENDUNG DES LÖSUNGSVORSCHLAGS 31
4.1 Kapitelübersicht 31
4.2 Erstellung einer Aufgabenaufzeichnung 31
4.3 Test Suite „Artikelanlage“ 33
4.3.1 Ermittlung der höchsten Artikelnummer 33
4.3.2 Artikel anlegen 37
4.3.3 Artikel prüfen 41

4.3.4 Artikel löschen 43
4.3.5 Produkt löschen 44
4.3.6 Vergleiche Dauer automatisierter Tests (AT) und manueller Tests 44
4.4 Testfall „Rohstoff bestellen“ 50
4.5 Testfall „Stückliste erstellen“ 53
4.6 Dokumentation und Speicherung in DevOps 57
4.7 Modultest „Einheitenumrechnung“ (UnitConversion_Test) 61
4.8 Modultest “Mehrwertsteuer ermitteln“ (VAT_Test) 63
4.9 Modultest „Artikel kopieren“ (ItemCopy_Test) 65
4.10 Vergleiche Dauer automatisierter Tests (AT) und manueller Tests - Modultests 67

5. ANALYSE DER ERGEBNISSE 72
5.1 Ergebnisse RSAT 72
5.2 Ergebnisse Modultests 74
5.3 Verwendete Techniken 75
5.4 Gewonnene Erkenntnisse 76

6. SCHLUSSFOLGERUNGEN 78
6.1 Verifikation der Hypothese 78
6.2 Beantwortung der Forschungsfrage 78
6.3 Zusammenfassung 79
6.4 Ausblick 79

LITERATURVERZEICHNIS 81

ABBILDUNGSVERZEICHNIS 83

TABELLENVERZEICHNIS 85

ABKÜRZUNGSVERZEICHNIS 86

1. EINLEITUNG

1.1 Ausgangssituation

Durch die Digitalisierung der produzierenden Industrie und dem Einsatz von Enterprise-Resource-Planning-Tools über die gesamte Wertschöpfungskette muss besonderes Augenmerk auf die Überprüfung der implementierten Funktionen, deren Korrektheit und die Stabilität des Systems gelegt werden.

Bis jetzt wurden bei der Firma Wewalka in jeder Abteilung ausgewählte Benutzer*innen damit beauftragt die neuen Funktionen zu testen. Diese Benutzer*innen sind aber nicht mit Softwareentwicklung oder Testmethoden vertraut, die implementierten Funktionen werden nur rudimentär auf Korrektheit überprüft. Problematisch sind Änderungen, die sich unbeabsichtigt auf andere Bereiche auswirken, die von den ausgewählten Benutzer*innen nicht überprüft werden beziehungsweise gar kein Zugriff besteht. Hier können sich Fehler verstecken, die im schlimmsten Fall erst nach Monaten an die Oberfläche treten. Weiters kann durch die ausgewählten Benutzer*innen in den Fachabteilungen bei jedem Plattform-Update nicht gewährleistet werden, dass diese Benutzer*innen sämtliche Funktionen und Formulare periodisch überprüfen.

Um eine effizientere Überprüfung zu gewährleisten werden die Testvorgänge automatisiert. Dadurch soll der Zeitaufwand gesenkt und die Mitarbeiter entlastet werden. Diese automatisierten Tests sollen in Zukunft bei jeder Funktionsimplementierung oder Update seitens des ERP-Herstellers durchlaufen, um etwaige Problematiken zu erkennen und Bereiche zu überprüfen, die unbeabsichtigt verändert wurden.

Die Forschungsfrage, welche ich mit dieser Arbeit beantworte lautet:

„Wird durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka der zeitliche Aufwand um den Faktor 5 verringert?“

Zusätzlich möchte ich mich um weitere Fragestellungen kümmern: Können manuelle Tests durch automatisierte Tests ersetzt werden? Sorgt eine automatische Überprüfung für ein stabileres System? In welchen Bereichen ist ein manueller Test gegenüber einer Automatisierung zu bevorzugen? Kann die Dokumentation der eingesetzten und durchgeführten automatischen Tests bei zukünftigen Anwendungsfällen hilfreich sein? Kann der automatische Test zu einer Kostenreduktion führen, da Mitarbeiter für komplexere Aufgabengebiete eingesetzt werden können?

1.2 Zielsetzung und Abgrenzung

Als langjähriger Softwaretester und Programmierer sind mir die Vor- und Nachteile von manuellem und automatisiertem Testen bestens bekannt. Durch die Umstellung auf ein automatisiertes Testsystem sollen einerseits die Mitarbeiter entlastet, andererseits ein stabileres System gewährleistet werden. Durch meinen Einblick ins das gesamte ERP-System soll durch Implementierung automatisierter Testvorgänge über die gesamte Supply Chain eine Kontrolle und Dokumentation der Daten und der Korrektheit der Funktionen entstehen.

Die Zielgruppen dieser Arbeit sind Entwickler*innen von ERP-Systemen - vorrangig Microsoft Dynamics 365 – und Software-Tester, die eine Überprüfung des eingesetzten Systems effizienter gestalten möchten. Für den Teil der Automatisierung, der ohne zusätzliche Implementierung erstellt wird, sind auch keine Programmierkenntnisse erforderlich. Weiters kann auch HR von den Erkenntnissen der Testautomatisierung profitieren, da Mitarbeiter*innen anderweitig eingesetzt werden können und keine Arbeitszeit für wiederkehrende Überprüfungen aufbringen müssten. Zu guter Letzt ist eine automatische Überprüfung der Systemstabilität und -korrektheit ein anzustrebender Zustand des gesamten wertschöpfenden Unternehmens.

Zusätzlich werde ich auch Subsysteme des ERP-Systems beleuchten, die mit automatisierten Tests schwer zugänglich sind und gegebenenfalls besser mit manuellen Tests abgedeckt werden sollten.

Nicht berücksichtigt werden Bereiche und Abteilungen, die mittels mobilem Scanner auf Android-Software Barcodes einlesen und in das ERP-System übertragen. Das verwendete Testautomatisierungstool läuft ausnahmslos auf Windows-Rechnern. Weiters könnte das eingesetzte Werkzeug zur vollständigen Automatisierung von Geschäftsprozessen dienen, diese Arbeit bezieht sich aber nur auf die Testautomatisierung, wobei auch die Testbereiche Benutzer*innenfreundlichkeit und GUI-Design von dem eingesetzten Tool nicht überprüft werden können.

Die Hypothese dieser Arbeit lautet:

„Durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka wird der zeitliche Aufwand um den Faktor 5 verringert.“

Auch wenn ich die Hypothese nicht beweisen kann, wird die Umstellung auf automatisierte Tests statt manuellen Tests von der Geschäftsleitung der Firma Wewalka gefordert und auch die Fachabteilungen stehen dieser Möglichkeit positiv gegenüber, von den Testpflichten befreit zu werden.

1.3 Persönliches Interesse

Da ich für mehr als sieben Jahre hauptberuflich als Software-Tester gearbeitet habe, bin ich mit den Aspekten des manuellen Testens bestens vertraut. So sind menschliche Fehler, wie auch „false positives“ bei Konzentrationsverlust oder monotonen und repetitiven Testaktivitäten immer möglich. Um diese Fehler zu minimieren fiel mein Augenmerk auf die Automatisierung der Testfälle. Als ich zu den Software-Entwicklern wechselte hatte ich die Ressourcen und die Umgebung um automatisierte Testfälle selbst zu erstellen.

Bei meinem jetzigen Arbeitgeber Wewalka wird der Bereich Software-Test derzeit nur nebenbei geführt, obwohl das System die gesamte Wertschöpfungskette abdeckt und ein kritischer Fehler sogar zu einem Produktionsstopp führen könnte, der enorm viel Kosten verursachen könnte.

Diese Risiken möchte ich im Unternehmen minimieren und den Arbeitskollegen Arbeit ersparen, indem ich diese Testautomatisierung des ERP-Systems implementiere.

1.4 Methodik und Aufbau

Für Dynamics 365 von Microsoft wird ein Programmiergerüst bereitgestellt, mit dem mittels Programmcode die Funktionen des ERP-Systems überprüft werden können. Dieses empfiehlt sich für einzelne Module oder abgrenzbare Programmeinheiten. Dieses Automatisierungswerkzeug wird zum Einsatz kommen, wenn eine neue, abgegrenzte Funktionalität in das ERP-System implementiert wird, die nicht mit bestehenden Modulen interagiert.

Für einen Testfall, der mehrere Ablaufschritte beinhaltet oder bei dem auf die Formularfelder und Menüführung zugegriffen werden muss, wird die Aufgabenaufzeichnung von Microsoft zum Einsatz kommen. Bei dieser Automatisierung kann eine Vorlage aufgenommen werden, die später mit anderen Parametern automatisch abgespielt werden kann.

Die manuellen Tests werden durch eine Bildschirmaufnahme und dessen integrierte Uhr gespeichert. Bei den automatischen Tests wird die Dauer des Durchlaufs notiert. Diese Zeiten werden am Ende gegenübergestellt.

1.5 Struktur der Arbeit

Der erste Teil der Arbeit befasst sich mit Hintergrundinformationen, den Ist-Zustand der Firma Wewalka, persönliche Interessen und das Ziel dieser Arbeit.

Im zweiten Kapitel werde ich die Grundlagen des automatisierten Software-Tests erörtern und wichtige Begriffe erklären, die zum Verständnis des Themas dienen. Weiters werde ich hier auch den Stand der Technik und den Stand der Wissenschaft niederschreiben.

In Kapitel Drei beschäftige ich mich mit dem konzeptionellen Vorgehens- und Lösungsansatz und dem Forschungsdesign, mit dem ich meine Hypothese bestätigen möchte. Weiters werde ich über die verwendeten Methoden, die Menge und Umfang der Testfälle und den Systemaufbau des ERP-Systems informieren.

2. GRUNDLAGEN UND DERZEITIGER STAND VON WISSENSCHAFT UND TECHNIK

2.1 Kapitelübersicht

Dieses Kapitel behandelt die Entstehung und Entwicklung des automatisierten Software-Tests und behandelt die Vor- und Nachteile von Tests durch menschliche Hand und programmierter Test-Scripts. Weiters werde ich auf fachspezifisches Vokabular eingehen, speziell auf jenes in der Microsoft-Umgebung und die derzeit eingesetzten Automatisierungstechniken beleuchten. Schlussendlich bietet dieses Kapitel den Aufbau und das Zusammenspiel der Entwicklungsmaschinen, des Beta-Systems und der Produktionsumgebung der Firma Wewalka.

2.2 Hintergrund automatisierter Tests

Im Laufe der Zeit wurden Software-Projekte und Programme immer komplexer und mussten in immer kürzeren Zeitabständen mit minimalen Ressourcen fertiggestellt werden. Um Kosten zu senken, jedoch ein qualitativ hochwertiges Produkt herzustellen, werden manuelle Testtätigkeiten durch automatisierte ersetzt (Dustin et al. 1999).

Durch den Einsatz von automatisierten Tests soll der Mensch als Fehlerquelle ausgeschlossen werden, da bei manuellen Tests immer die Gefahr besteht, dass Werte falsch eingegeben oder abgelesen, „false positives“ notiert werden oder die Testqualität durch Übermüdung oder Unkonzentriertheit leidet.

Die Automatisierung hat zusätzlich den Vorteil, dass sie bei korrekten Testergebnissen bei jeder zukünftigen Iteration der Software erneut eingesetzt werden kann. So entsteht mit der Zeit eine Datenbank an Testfällen, die zunehmend für ein stabiles System sorgt und etwaige Fehlverhalten aufdeckt. Weiters kann die bestehende Automatisierung außerhalb der Geschäftszeiten getätigt werden, es wird für den Durchlauf keine menschliche Interaktion benötigt. Da bei automatisierten Tests die visuelle Darstellung des Systems oftmals nicht oder nur spärlich angezeigt wird können diese die Durchlaufzeit enorm verkürzen, da viel Systemzeit durch Anzeige von visuellen Inhalten oder durch Warten auf Benutzer*inneneingaben generiert wird.

Der größte Aufwand liegt bei der Erstellung und Programmierung von automatisierten Testfällen und -skripten. Die Einholung der bisher verwendeten Testfälle bis zur Implementierung in das Automatisierungssystem ist mit viel Zeit- und Ressourcenaufwand verbunden.

Ein weiterer Nachteil der Testautomatisierung besteht darin, dass für die Skripterstellung Programmierkenntnisse unbedingt erforderlich sind. Bei manuellen Testdurchläufen oder bei der Aufgabenaufzeichnung werden diese Kenntnisse hingegen nicht benötigt. Weiters müssen auch bei größeren Softwareänderungen automatisierte Testfälle angepasst werden. Sollte beispielsweise eine Funktion aus dem Code gelöscht werden, müssen auch die Testfälle umgeschrieben werden, die bislang auf diese Funktion zugegriffen haben. Aber auch Testfälle, die ohne Programmieraufwand aufgezeichnet wurden müssen kontrolliert werden, wenn ein Steuerelement auf der Eingabemaske entfernt wurde. Die Automatisierung würde in diesem Fall fehlschlagen, da sie dieses Element nicht mehr finden würde.

Die folgenden Teilbereiche des Software-Tests können durch eine Automatisierung abgebildet werden (Dustin et al. 2001):

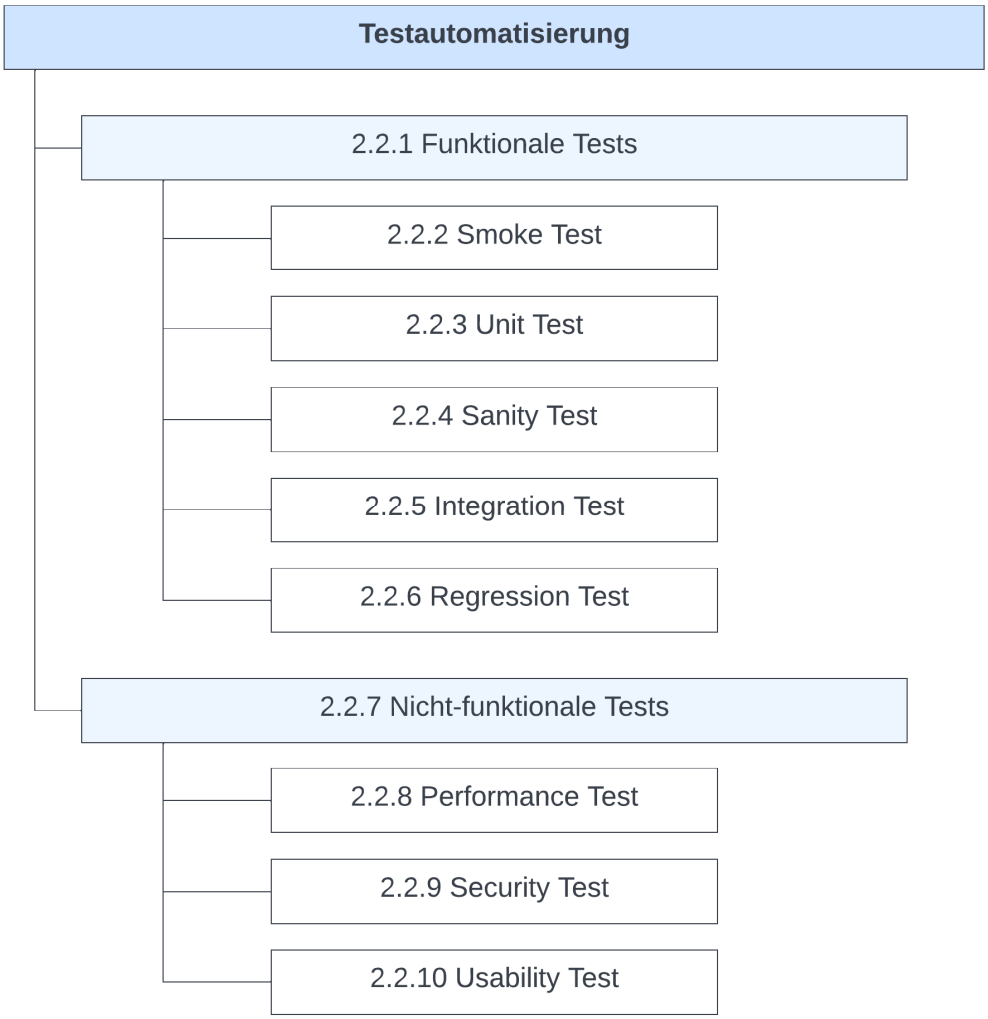

Abbildung 1: Abgedeckte Teilbereiche des Software-Tests. (eigene Abbildung in Anlehnung an Dustin et al. 2001)

2.2.1 Funktionale Tests

Bei den funktionalen Tests wird das System, Module und ihr Zusammenspiel oder einzelne Funktionen auf die korrekte Funktionsweise geprüft. Durch das Gegenüberstellen der vorgegebenen Spezifikationen mit dem Output des Testobjekts kann die korrekte Funktionalität abgebildet werden.

2.2.2 Smoke Test

Grundlegende Funktionen des Systems werden beim Smoke Tests oberflächlich überprüft. Er ist als erstes bei einer neuen Programmversion durchzuführen, es werden beispielsweise alle Bedienelemente auf Funktionalität überprüft. Sollte diese fehlschlagen, brauchen komplexere Methoden und Funktionen nicht weiter getestet werden, da selbst die Grundfunktionalität nicht gegeben ist. Bei einer Testautomatisierung geschehen diese Smoke Tests in abgekapselten Unit Tests nach der Kompilation der neuen Programmversion.

2.2.3 Unit Test

Einzelne Module oder Funktionen werden auf ihre ordnungsgemäße Implementierung überprüft. Dies wird in der Regel von den Entwicklern selbst übernommen, da hier Programmierkenntnisse und Kenntnisse der verwendeten Logik und des internen Systems benötigt werden.

2.2.4 Sanity Test

Bei Updates, Patches oder Bugfixes wird die ausgebesserte Version einem Sanity Check unterzogen. Es wird geprüft, ob die Codeänderungen im System den vorher aufgedeckten Fehler ausbessert und keine neuen fehlerhaften Funktionalitäten entstanden sind. Automatisierte Test-Skripten und Testfälle müssen natürlich bei inkorrektem Verhalten oder Ergebnissen auch ausgebessert, verbessert und einem Sanity Test unterzogen werden.

2.2.5 Integration Test

Der Integrationstest dient dazu, die im Unit Test getesteten Einzelmodule in ihrem Zusammenspiel und ihren Verbindungen zueinander miteinander zu testen. Es werden die zusammengehörigen Funktionen und Schnittstellen der zusammenhängenden Module überprüft.

2.2.6 Regression Test

Durch Änderungen, Patches, Bugfixes im System können unbeabsichtigt Probleme in Bereichen der Software entstehen, die von Programmierer*innen übersehen oder nicht bedacht wurden. Beim Regressionstest werden vorher erstellte und geprüfte Testfälle

durchgeführt, die die Stabilität des Gesamtsystems und die weitere korrekte Ausführung nicht geänderter Bereiche sicherstellen sollen.

2.2.7 Nicht-funktionale Tests

Bei den nicht-funktionalen Tests werden Bereiche überprüft, die nicht die Primärfunktionen des Systems oder der Software abdecken. Diese beschäftigen sich mit dem „Look and Feel“, der Performance und der Sicherheit der Anwendung.

2.2.8 Performance Test

Beim Performance Test soll das Verhalten des Systems bei Normallast, Stresslast und Minimallast überprüft werden. Der Performance Test soll bei einem funktional stabilen System das Verhalten der Anwendung unter Auslastung überprüfen. Bei der Lastüberprüfung beispielweise werden entweder viele Funktion schnell hintereinander ausgeführt oder viele Benutzer*innen starten parallel Aktivitäten. Somit kann ermittelt werden, ob das System unter Last noch bedienbar ist, ob die getesteten Funktionen unter Last korrekt arbeiten und ob sich das System nach einer Lastphase wieder in den Normalzustand stabilisieren kann. Eine Automatisierung ist hier hilfreich, da beispielsweise viele virtuelle User parallel simuliert werden können. Zeitgleich kann der Performance Test auch auf schwache Hardware hinweisen, die für den Betrieb des Systems nicht ausreichend ist.

2.2.9 Security Test

Damit Anwender*innen nur Bereiche einsehen und bearbeiten können, die für ihren Bereich notwendig sind, müssen Zugriffsregelungen getroffen werden. Auch Unbefugten oder betriebsfremden Personen muss der Zugang zum System und der Zugriff auf Daten verwehrt werden. Um dies zu gewährleisten muss ein Zugriffssystem mit spezifizierten Rollen, Rechten und Privilegien installiert werden. Um die Vielzahl der verschiedenen Benutzergruppen zu überprüfen und einen unrechtmäßigen Zugriff zu testen kann eine Automatisierung die verschiedenen Zutrittslevel überprüfen und rückmelden, ob eine unbefugte Gruppe Zugriff zu einem geschützten Bereich hat.

2.2.10 Usability Test

Hier erfolgt die Automatisierung meist durch eine Aufzeichnung von Benutzer*inneneingaben in der Anwendung. Es kann analysiert werden, ob der User ohne Fehlklicks eine vorgegebene Aufgabe in einer angemessenen Zeit erfüllen kann. Durch die Auswertung der Testergebnisse können Schwachstellen in der Benutzbarkeit ermittelt werden, die zu einem Re-Design des betroffenen Moduls führen können.

Durch eine Abdeckung der einzelnen Stufen mit Automatisierungen sind die Stufen Systemtest und User Acceptance Test in Abbildung 1 nicht mitangeführt und werden hier nur der Vollständigkeit halber angeführt:

2.2.11 System Test

Das Endergebnis der einzelnen Teststufen bildet der System Test: Hier werden alle funktionalen und nicht-funktionalen Tests zusammengeführt und das komplette System auf die gegebenen Anforderungen geprüft.

Eine zukünftige Möglichkeit wäre nach der Ausrollung ins Produktiv-System ausgewählte Tests automatisiert ablaufen zu lassen um eine stichprobenartige Überprüfung des Build-Systems vorzunehmen und die Performanz des Produktiv-Systems nach Implementierung neuer Funktionen zu testen.

2.2.12 User Acceptance Test

Der User Acceptance Test oder Abnahmetest ist die Überprüfung, ob die Anwendung wie beschrieben funktioniert und auch bedienbar ist – aus Sicht der Benutzer*innen. Die Anforderungen werden bei Wewalka von den Benutzer*innen selbst gestellt und deren Realisierung weiterhin manuell auf dem Staging-Server überprüft. Diese Staging-Plattform ist die Zwischenstufe zwischen Entwicklung und Produktiv-System, hier können die Benutzer die neuen Funktionen und Ausbesserungen überprüfen („Beta-System“).

2.3 Continuous Integration (CI)

Bei der Continuous Integration werden die Codeänderungen und Neuentwicklungen aller Programmierer*innen des Projekts regelmäßig in einem Repository zusammengefasst. Diese Änderungen werden in einem Version Control System (Versionsverwaltung) versioniert und abgespeichert. Der CI-Server erkennt diese und startet ein Build-Script, welches die neueste Programmversion erstellt. Weiters ist der CI-Server verantwortlich, den Entwickler*innen Feedback über den Build-Vorgang mitzuteilen, wie in Abbildung 2 ersichtlich.

Von Vorteil ist hier ein eigener Build-Server, der eigens konfiguriert und mit spezifischen oder allen automatisierten Testvorgängen gefüllt werden kann (Fowler 2006). Zwar ist ein kompletter Durchlauf der Testautomatisierung erstrebenswert, bei kritischen Auslieferungen kann aber ein Großteil deaktiviert werden, der den Build-Vorgang um Stunden verkürzt.

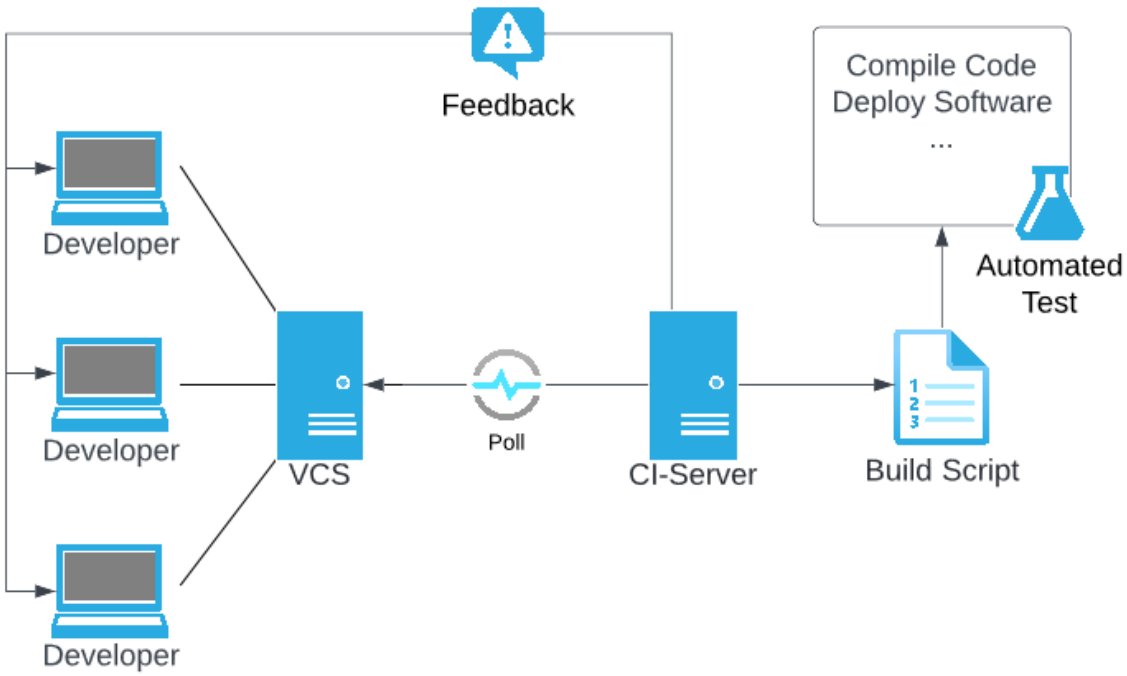

Abbildung 2: Continuous Integration mit Build Process. (eigene Abbildung mit Anlehnung an Duvall et al, 2007)

Bei der Firma Wewalka wird die Ausrollung eines erfolgreichen Build-Vorgangs aber anfänglich nicht auf das Produktiv-System (Live System) vorgenommen, sondern auf einen Staging-Server, auf dem ausgewählte Benutzer*innen ihre Überprüfungen vornehmen. So wird sichergestellt, dass Programmänderung keine kritischen Systemabstürze produzieren, die am Produktiv-System im schlimmsten Fall einen Stopp der Produktion zur Folge hätten. Da die Firma im Mehrschichtsystem rund um die Uhr

arbeitet, ist es auch weiters nicht möglich das Produktiv-System für mehrere Stunden abzuschalten, um einen kompletten automatisierten Testlauf durchzuführen. Deswegen werden auch die automatisierten Tests bei den Builds für den Staging-Server aktiviert werden. Nach erfolgreicher Testausführung wird der Softwarestand vom Staging-Server in das Produktivsystem geklont.

Während der Einführung der automatischen Testüberprüfung wird bei Wewalka ein zweiter Build-Server aktiviert, der die Testausführung vornimmt, am Ende aber keine Software-Pakete schnürt. Die Erstellung der Pakete übernimmt weiterhin der erste stabile Build-Server, der von den Änderungen und Automatisierungen unberührt bleibt. Dies dient als Absicherung, falls es Komplikationen am Test-Build-Server geben sollte. Durch diese Maßnahme können bei Problemen mit den automatisierten Tests weiterhin neue Software-Versionen und Bugfixes in das Produktiv-System eingespielt werden.

Durch eine Kopie der echten Datenbank kann auch die Staging-Umgebung mit reellen Daten versehen und die automatisierten Tests mit diesen arbeiten. Diese Datenkopie erfolgt im Hintergrund und beeinflusst das Produktivsystem nur durch minimaler zusätzlicher Last.

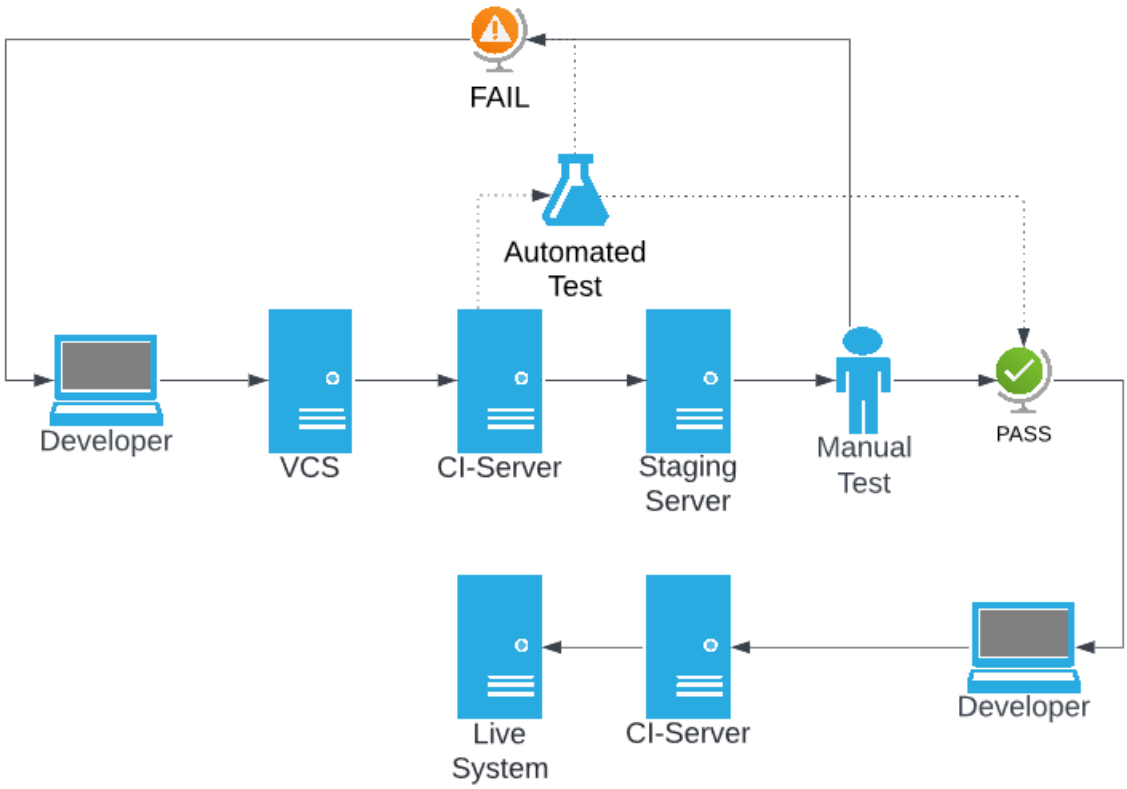

Abbildung 3: Deployment und Test bei Wewalka (eigene Abbildung, 2022)

2.4 Aktueller Stand der Technik

Grundsätzlich unterscheiden sich die Arten der Testautomatisierung aufgrund ihres Einsatzgebietes. Während in der modularen Ebene noch Programmierkenntnisse erforderlich sind und die Testfälle in Programmcode entwickelt werden, werden automatisierte Testfälle bei Integrationstests mit simulierten Peripherie-Geräten und -eingaben verwirklicht. Automatisierungs-Tools bei nicht-funktionalen Tests, wie Performance-Tests, arbeiten mit Kennzahlen, um beispielsweise die gewünschte Antwortzeit zu überprüfen.

[Grafik-Überblick über die 10 Arten mit Kapitelnummer]

Folgend ist eine Auflistung der derzeit verbreitetsten Modul-Automatisierungs-Tools in .NET-Umgebungen für die Programmiersprache C#:

2.4.1 MSTestV2

Microsoft Unit Test Framework ist ein in die Entwicklungsumgebung Visual Studio integrierter Modultest. Für erstellte Methoden kann deren Funktionalität und Korrektheit durch programmierte Testmethoden überprüft werden.

Ein Beispiel zur Überprüfung einer Methode, die zwei positive Zahlen addiert, aber bei einer negativen Zahl -1 zurückliefert:

```
public int PosNumAdd (int num1, int num2)
{
    if (num1 < 0 || num2 < 0)
        return -1;
    return num1 + num2;
}

[TestMethod()]
public void PosNumAddTest()
{
    int result = PosNumAdd(10, 30);
    Assert.IsTrue(result == 40);
}
```

Durch das Attribut [TestMethod()] wird die nachfolgende Methode automatisch in den Testkatalog aufgenommen und kann danach entweder lokal gestartet oder danach im Build-Prozess automatisch bei Paketerstellung ausgeführt werden.

Mit *Assert* wird der zurückgegebene Wert überprüft, in diesem Fall wird mit *IsTrue* abgefragt, ob die Variable *result* 40 gleicht. Dies deckt zwar nur einen Fall ab, bestätigt aber die korrekte Funktionalität der Funktion *PosNumAdd* für die ausgewählten Werte.

2016 wurde der Vorgänger MSTestV1 durch MSTestV2 ersetzt. Der Nachfolger wurde auf open-source umgestellt und unterstützt Windows, Linux und Mac.

2.4.2 NUnit

NUnit ist eine quelloffene Test-Umgebung, die von JUnit (Modultest für Java-Applikationen) hervorgegangen ist. Sie wurde über 126 Millionen Mal heruntergeladen (Stand: März 2021). Die Testautomatisation kann über die Benutzeroberfläche in Visual Studio oder über ein CLI (Command Line Interface) gestartet werden. Test können hier hintereinander oder parallel ausgeführt werden.

```
public int PosNumAdd (int num1, int num2)
{
    if (num1 < 0 || num2 < 0)
        return -1;
    return num1 + num2;
}

[Test, Order(1)]
public void PosNumAddTest()
{
    int result = PosNumAdd(10, 30);
    Assert.IsTrue(result == 40);
}
```

2.4.3 xUnit.Net

Ist eine quelloffene Testumgebung in der .NET-Umgebung. Der größte Unterschied zu MSTestV2 und NUnit ist die Isolation der einzelnen Tests. Die Testfälle werden gestartet, ausgeführt und danach sofort verworfen. Das garantiert, dass die Testfälle in beliebiger Reihenfolge ausgeführt werden können und minimiert die Gefahr, dass ein Testfall andere folgende Testfälle scheitern lässt.

```
public int PosNumAdd (int num1, int num2)
{
    if (num1 < 0 || num2 < 0)
        return -1;
    return num1 + num2;
}
```

```
[Fact]
public void PosNumAddTest()
{
    int result = PosNumAdd(10, 30);
    Assert.IsTrue(result == 40);
}
```

Weiters ist xUnit.Net die Umgebung, die am meisten von der Community profitiert und weiterentwickelt wird. Sie ist die einfachste erweiterbare Umgebung. Zusätzlich bietet xUnit.Net im Gegensatz zu MSTestV2 und NUnit mit *Assert.Throws* eine generelle Fehlerüberprüfung an. Bei den zwei anderen Umgebungen müssen Fehler spezifisch abgefangen werden, bei xUnit.Net schlägt mit *Assert.Throws* der Testfall auch fehl, auch wenn der Fehler nicht spezifisch abgefangen wurde.

2.4.4 Entscheidung Modultest-Umgebung

Zwar sind die drei beschriebenen Umgebungen gleichwertig und haben keine gravierenden Vor- oder Nachteile, doch habe ich mich bei der Firma Wewalka für MSTestV2 entschieden. Diese Umgebung bietet die sauberste Programmierung und verfügt über eine umfassende Dokumentation über Erstellung der Testfälle, wie auch Einbindung in den Build-Prozess in Microsofts Azure DevOps.

Alle drei Umgebungen verfügen über eine Einbindung des Selenium Webdrivers, der aber erst mit dem Regression Suite Automation Tool beim Integration Test in die Testautomatisierung eingebunden wird. Die Modultests benötigen keinen Kontakt zum Browser, diese arbeiten isoliert in den Methoden und Datenbanken.

Im Bereich Integration Test wird das Zusammenspiel der einzelnen Module und deren Korrektheit überprüft. Hier finden die Testfälle auf der Oberfläche der Software mit simulierten Peripherie-Eingaben statt. Im Falle von Microsofts D365 – Finance & Operations passiert dies im Browser. Firma Wewalka schreibt hier Microsoft Edge als Browser für das ERP-System vor. Durch die Wiederholbarkeit der Testfälle werden diese in einer Datenbank aufgenommen und können von nun an auch bei Versionsupdates bei Regressionstests ausgeführt werden.

Folgend eine Auflistung von Automatisierungs-Tools, die für den Integrationstest und Regressionstest eingesetzt werden können:

2.4.5 Selenium

Selenium ist das Gerüst vieler Automatisierungsumgebungen und passiert auf HTML und JavaScript. Während die Benutzer*innen nur mit dem Browser interagieren, zeichnet Selenium die verwendeten Formularelemente und Werte auf. Diese können später beliebig oft wiederholt werden („Capture-Replay“).

Das seit Oktober 2016 verfügbare Framework Selenium 3 besteht aus folgenden Komponenten:

- Selenium Core
  Enthält die Testkommando-API und den ausführbaren TestRunner.
- Selenium IDE
  Ist als PlugIn für die Browser Mozilla Firefox und Google Chrome verfügbar. Durch die Verwendung des Browsers durch die Benutzer*innen können Testfälle aufgezeichnet werden und danach durch die Selenium DIE abgespielt und überprüft werden.
- Selenium Webdriver
  Ermöglicht die Testautomatisierung von grafischen Benutzeroberflächen, basierend auf verschiedenen Programmiersprachen, wie Java, C#, Perl, PHP und Ruby.
- Selenium Grid
  Ermöglicht die parallele Ausführung von Testfällen auf mehreren Servern, um die Dauer der Überprüfung zu minimieren.

2.4.6 Appium

Appium ist eine quelloffene Automatisierungsumgebung, die primär für iOS-, Windows- und Android-Apps entwickelt wurde. Sie läuft auf dem WebDriver-Protokoll und kann mit verschiedenen Programmiersprachen, wie Ruby, Java, JavaScript, Python und PHP verwendet werden.

2.4.7 Protractor

Dient als Automatisierungsumgebung für Web-Anwendungen, die mittels JavaScript oder TypeScript verwirklicht wurden. ProTractor ist ein Wrapper für den Selenium Webdriver und spezialisiert auf die Überprüfung von AngularJS-Anwendungen.

2.4.8 SikuliX

SikuliX ist ein Automatisierungstool, um grafische Benutzeroberflächen mittels Screenshots zu testen. Durch Bilderkennungs-Algorithmen kann SikuliX mit Formularfeldern auf der Website interagieren, hierbei interagiert SikuliX mit dem Selenium Webdriver.

2.4.9 Tricentis Tosca

Das von der Firma Tricentis vertriebene Automatisierungs-Tool Tosca verfügt zusätzlich über ein integriertes Testmanagement, eine grafische Benutzeroberfläche und eine API. Unterstützt werden die Programmiersprachen C#, C++, Visual Basic 6 und Java auf Microsoft Windows Plattformen.

Zusätzlich verfügt das Tool über eine zertifizierte SAP-Anbindung für automatisierte Tests des ERP-Systems.

Tosca wird als proprietäres System angeboten und wird beispielsweise von der Deutschen Börse, EVN AG und OMV eingesetzt.

2.4.10 Regression Suite Automation Tool (RSAT)

Das kostenlose Automatisierungstool RSAT von Microsoft ist für den Betrieb von D365 – Finance & Operations ausgelegt. Die Grundlage der Testfälle bildet die Aufgabenaufzeichnung des ERP-Systems. Bei dieser werden Testfälle mit einzelnen Schritten durchgearbeitet, die Aufzeichnung speichert die verwendeten Formulare, Felder und Werte ab. Diese Aufzeichnung wird dann in RSAT eingefügt und die verwendeten Werte in einem Microsoft-Excel-File als Parameter abgespeichert. Diese Parameter-Files können verändert werden, um die durchzuführenden Tests mit dynamischen Werten abzuarbeiten.

RSAT beruht auf dem Webdriver und kann mittels Microsoft Edge und Google Chrome eingesetzt werden. Zusätzlich kann das Tool mit Azure DevOps kombiniert werden, so werden alle durchgeführten Tests protokolliert und deren Ergebnisse und Durchgänge dokumentiert.

Die Parameter und Ergebnisse von Testfällen können einem folgendem Testfall übergeben werden, sodass man einen dynamischen End-to-End-Test verwirklichen kann.

Weiters können in den Testfällen Systeminformationen, wie Pop-Ups, überprüft werden. Beispielsweise könnte überprüft werden, ob eine Warnmeldung ausgegeben wird, wenn ein Kunde angelegt wird, der bereits im System existiert.

Das Security Testing kann von RSAT übernommen werden, so kann man jeden Testlauf mit einem spezifischen User vornehmen, also überprüfen, welche Usergruppen zu welchen Formularen Zugriff hat und ob das System einen Fehler meldet, wenn nicht-authorisierte Benuzer*innen in einen gesperrten Bereich einsteigen wollen.

Um bei der CI die Tests automatisch ablaufen zu lassen, muss das RSAT-Tool auf dem Build-Server installiert werden, der den Build-Prozess abarbeitet. Nach erfolgreicher Kompilation der Software werden die aktivierten Testfälle durchgeführt.

2.4.11 Entscheidung Integrationstest-Umgebung

Durch die Verbreitung von Selenium und die Einbindung von RSAT in die gesamte Entwicklungsumgebung mit gleichzeitiger Dokumentation habe ich mich für das Regression Suite Automation Tool von Microsoft für die Testautomatisierung entschieden. Protractor kann in Zukunft für die mobilen Scanner-Apps im Betrieb in

Erwägung gezogen werden. SikuliX ist durch das Überprüfen durch Screenshots weniger performant als der Selenium Webdriver, Tricentis Tosca hat Erfahrungen im ERP-Bereich, verfügt über einen Support-Bereich, liegt aber außerhalb des Budgets.

2.5 Aktueller Stand der Wissenschaft

Um eine höchstmögliche Fehlerlosigkeit von Software zu garantieren ist der Software-Test der am weitest verbreitete Ansatz zur Qualitätssicherung. Diese Kosten betrugen Anfang 2000 zwischen 30% und 40% des Entwicklungsbudgets der beobachteten Projekte (Pol et al. 2002) und steigerten sich zehn Jahre später auf 30% bis 60% des Entwicklungsbudgets der untersuchten Projekte (Ebert 2011).

Durch einen Wandel in der Software-Entwicklung zu agilen Methoden, dem testgetriebenen Ansatz und kürzeren Intervallen zwischen Software-Releases müssen immer mehr Tests in immer kürzer werdenden Zeiten erledigt werden.

2.5.1 Agile Software-Entwicklung

Im Februar 2001 entstand in Utah das agile Manifest, um Werte und Prinzipen der agilen Softwareentwicklung festzulegen. Durch diesen agilen Prozess wurden die Entwicklungsiterationen kürzer, funktionierende Softwarestände mussten in kürzeren Abständen an Kund*innen ausgeliefert und Anforderungsänderungen und Zusatzfunktionalität dynamisch umgesetzt werden.

Die Prinzipien der agilen Softwareentwicklungen lauten wie folgt (Beck et al. 2001):

- *Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.*
- *Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.*
- *Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.*
- *Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.*
- *Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.*
- *Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.*
- *Funktionierende Software ist das wichtigste Fortschrittsmaß.*

- *Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.*
- *Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.*
- *Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.*
- *Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.*
- *In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.*

Mindestens vier der zwölf Prinzipien beziehen sich auf Software-Funktionalität, -Änderungen und Fehlervermeidung, um den Kund*innen eine höchstmögliche Qualität der Software zu liefern. Durch diese agilen Methoden konnte der Software-Test nicht mehr nur am Ende des Entwicklungsprozesses existieren, er musste in jeder Iteration durchlaufen werden und war gleichzeitig auch für die Überprüfung von Zusatzfunktionen und Anforderungsänderungen verantwortlich. Dieser Mehraufwand und Wiederholungen von bereits durchgeführten Tests wurde immer öfter durch eine Testautomatisierung realisiert.

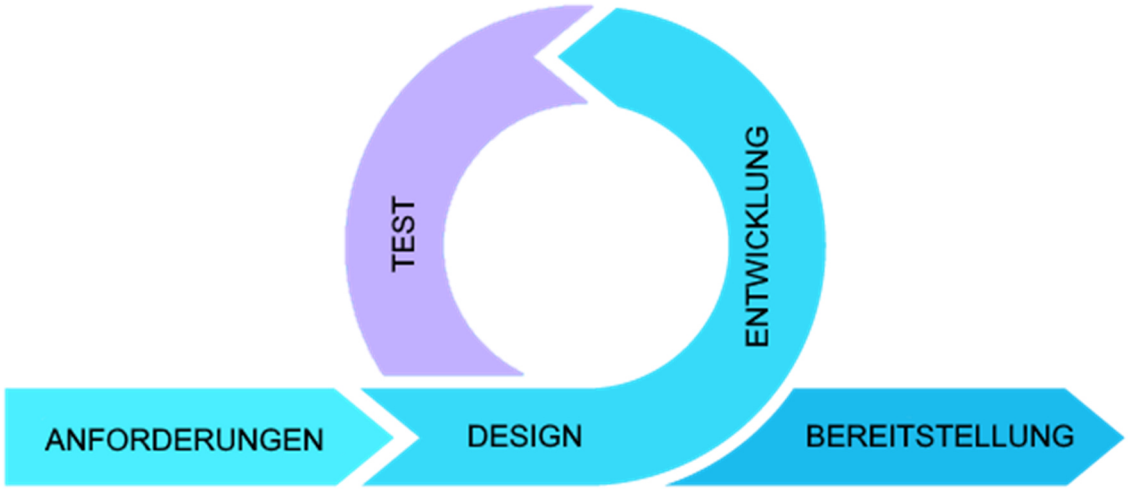

Abbildung 4: Test in jeder Iteration der agilen Software-Entwicklung. (eigene Abbildung, 2022)

Abbildung 4 zeigt den iterativen Ablauf des Projekts, ausgehend von den Anforderungen wird die Design-, Entwicklungs-, und Testphase bei jeder Iteration wiederholt. Nach erfolgreichem Abschluss wird die Software bereitgestellt.

Zwar sind die Erstkosten bei einer Testautomatisierung weit höher als bei manuell durchgeführten Tests, jedoch sinken die laufenden Kosten bei steigenden Entwicklungszyklen und durchgeführten Testfällen. Funktioniert beispielsweise ein

automatisierter Test ordnungsgemäß kann dieser durch seine Wiederverwendbarkeit bei den folgenden Zyklen erneut eingesetzt werden. Im Gegensatz dazu würde ein manueller Test mehr Zeit in Anspruch nehmen, außerdem wäre bei jedem manuell durchgeführten Testdurchlauf die Gefahr eines menschlichen Fehlers gegeben.

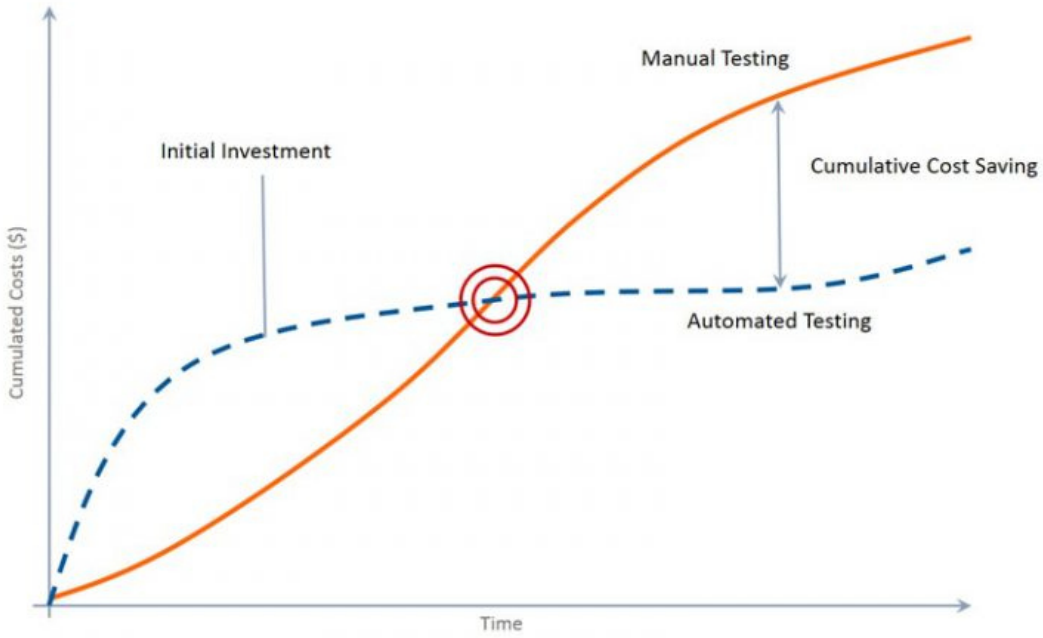

Abbildung 5: Verlauf der Kostenkurven bei Ausführung von manuellen und automatischen Tests (Palamarchuk, 2015)

Abbildung 5 zeigt die höheren Kosten der Testautomatisierung bei der Einführung, hier muss das Testsystem, die Test-Tools erst angeschafft und implementiert und Testfälle erzeugt werden. Mit fortschreitender Dauer (oder Zyklen) werden aber bereits erzeugte Testfälle verwendet und die Testautomatisierung benötigt weniger Instandhaltungskosten als die Durchführung manueller Tests.

2.5.2 Test Driven Development (TDD)

Die testgetriebene Entwicklung ist ein Ansatz, der bei agiler Software-Entwicklung angewendet wird. Hier programmieren die Entwickler gleichzeitig die Software-Tests und die zu testende Komponente.

Module und deren Modul-Tests werden parallel zueinander entwickelt, wobei nicht festgelegt ist, dass der Programmierer des Moduls auch deren Tests entwickelt. Die eigentliche Programmierung ist in Iterationen unterteilt, die nur wenige Minuten dauern sollten.

Diese Iterationen werden Red-Green-Refactoring benannt (Beck, 2002):

Abbildung 6: Red-Green-Factoring (eigene Abbildung, 2022)

Abbildung 6 zeigt das Red-Green-Factoring von Beck. Im Folgenden wird diese Methode detaillierter beschrieben:

Im ersten Schritt (Red) wird ein Test erstellt, der eine neue Funktionalität prüfen soll, die aber noch nicht implementiert wurde. Ohne eine passende Implementierung schlägt der Test hier fehl.

Aufbauend darauf wird mit möglichst wenig Aufwand der Programmcode abgeändert oder hinzugefügt (Green), sodass alle Testfälle erfolgreich abschließen.

Im dritten Teil wird der Code bereinigt und an Konventionen angepasst, um einen schlichten und lesbaren Programmcode zu erhalten.

Ist die Funktionalität der Software gewährleistet und alle Testfälle erfolgreich abgeschlossen ist das getestete Modul fertig. Die erstellten Testfälle werden nun gespeichert für zukünftige Regressionstest bei etwaigen Software-Änderungen.

2.5.3 Entscheidungsgrundlagen zur Testautomatisierung

Die auch heute noch geltenden Prinzipien für die Einführung und den Betrieb von Testautomatisierung fasse ich nach den Erkenntnissen von Polo et. al mit eigenen Worten zusammen: (Polo et al. 2013):

- Die Auswahl der Test-Tools muss gewissenhaft erfolgen. Diese können nicht bei jedem neuen Projekt ausgetauscht werden, weiters binden professionelle Umgebungen von Drittanbietern einen hohen Kapitalwert.
- Für jede Testaktivität müssen klare und messbare Qualitätsziele gesetzt und Anfangs- und Endkonditionen definiert werden.
- Ständige Messung und Verbesserung der Testeffizienz und der Testeffektivität.
  Testeffizienz: Welche Ressourcen müssen aufgewendet werden, um einen Fehler zu detektieren.
  Testeffektivität: Wie viele Fehler können mit der eingesetzten Methodik detektiert werden und welcher Bereich der möglichen Fehlereingaben durch den Test abgedeckt werden.
- Testfälle müssen so spezifisch wie möglich sein. So wäre ein abstrakter Testfall, der eine Vielzahl von Szenarien abdeckt wenig aussagend. Hier würde sich eine Testsammlung anbieten, in der einzelne Testfälle auf einzelne Szenarien eingehen der bessere Weg.

- Die Aussagen, ob ein Testfall bestanden hat oder fehlgeschlagen ist müssen vorsichtig gewählt werden. So können durch „false positives“ Fehler in der Software übersehen werden, die durch die Automatisierung als korrekt empfunden wurden
- Kontaminierung von Testfällen sind zu vermeiden – Ergebnisse eines Testfalls dürfen andere Testfälle nicht beeinflussen – Ein fehlerhafter Output eines Tests könnte alle folgenden Tests zum Scheitern bringen. Eine Ausnahme in einer Testsequenz mit Testfällen, die aufeinander aufbauen, wäre der sofortige Abbruch bei einem fehlerhaften Test. So wird die Abarbeitung unterbrochen und fälschlicherweise falsch gemeldete Tests werden nicht weiter ausgeführt.
- Ausgewogene Lastverteilung während der Testausführung – Gerade bei nicht-funktionalen Tests wie dem Performance-Test können durch falsche Lastverteilung korrupte Werte zurückgeliefert werden. Bei Stresslast-Tests hingegen sollte die Last auf einen Punkt konzentriert werden.
- Die Integrität der Test Suite muss gewährleistet sein. Durch Wartung und gegebenenfalls Anpassungen am Automatisierungstool und den Testfällen müssen diese regelmäßig auf Korrektheit und weiterhin bestehende Funktionalität überprüft werden.

Zusätzlich zu diesen Prinzipien wird das Augenmerk auf weitere Notwendigkeiten und Anforderungen gelegt:

- Die Ausführung und Ergebnisse der automatisierten Testfälle muss ständig kontrolliert werden, um die Integrität der Testfälle zu gewährleisten.
- Die Speicherung der Testfälle und deren Resultate müssen in einer zentralen Umgebung geschehen. Werden die Testergebnisse beispielsweise nur lokal auf den jeweiligen Entwicklerrechnern gespeichert, wird die Rückverfolgung eines Fehlers erschwert oder sogar unmöglich gemacht. Durch zentrale Speicherung kann auf einen Blick kontrolliert werden, wieso und ab welchem Zeitpunkt ein Testfall fehlschlägt.
- Der Zugriff auf Testfälle und Testergebnisse muss durch ein Zugriffskontrollsystem gewährleistet werden. So sollen automatische Tests nur gestartet werden, wenn die Software bereits in der Testphase ist und nur von Benutzern gestartet werden, die über Wissen der Testfälle besitzen. Unbefugte könnten hier Inkonsistenzen bei den Testergebnissen erzeugen oder Ressourcen binden, die zu diesem Zeitpunkt nicht vorgesehen waren. Falsche Testergebnisse könnten die Entwickler zu einer Fehlersuche bewegen, obwohl gar kein Fehler zu diesem Zeitpunkt existiert.
- Überprüfung der Reihenfolge der Testfälle und deren Synchronisierung. In Testsequenzen bauen oftmals Testfälle auf vorangegangenen Testfällen auf und

bedienen sich ihrer Output-Variablen. Eine falsche Anordnung würde einen oder mehrere Testfälle als fehlgeschlagen ansehen.

- Kontrolle von gemeinsam verwendeten Daten und Variablen. Daten, die von mehreren Testfällen verwendet werden dürfen nicht ohne Absprache verändert werden. Ein geänderter Testfall könnte durch geänderte Daten andere Testfälle ungewollt fehlschlagen lassen.
- Gegebenenfalls müssen Testresultate von verschiedenen Test-Teams auf eine gemeinsame Basis gebracht werden. Die unterschiedliche Repräsentation der Daten muss in ein einheitliches Datendesign transportiert werden, um eine transparente Sicht zu ermöglichen.
- Dokumentation über automatisierte Testfälle und manuelle Testfälle. Um Missverständnisse vorzubeugen hat die Erfahrung gezeigt, dass es sinnvoll ist, zu dokumentieren, welche Tests weiterhin händisch ausgeführt werden und auch diese Testergebnisse in der zentralen Umgebung neben den automatisierten Tests zu speichern. So werden Unklarheiten, ob manche Gebiete oder Funktionen nicht getestet wurden, beseitigt.

2.5.4 Hindernisse bei der Testautomatisierung

Die Implementation einer Testautomatisierung neben den manuellen Tests ist selbst ein kritisches Projekt, welches durch zahlreiche Hürden erschwert werden kann (Wiklund et al. 2017):

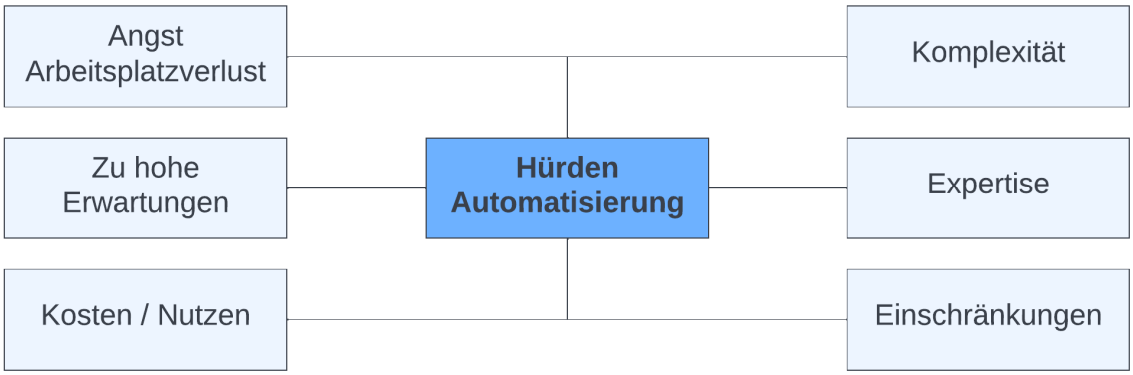

Abbildung 7: Hürden bei der Einführung von Testautomatisierung. (eigene Abbildung in Anlehnung an Wiklund et al., 2017)

Abbildung 7 beschreibt die Hürden, die bei der Implementierung der Testautomatisierung auftreten können:

Fast schon historisch bedingt ist die Angst vor dem Arbeitsplatzverlust bei Einführung neuer Automatisierungen. Hier muss die Unterstützung vom Management gegeben sein um die Motivation der Mitarbeiter auch für die Automatisierung hoch zu halten. Denn eine Testautomatisierung hat nicht das Ziel Mitarbeiter abzubauen, sondern diese für

kritischere und komplexere Aktivitäten freizubekommen. Eine Miteinbindung der betroffenen Mitarbeiter und eine Identifikation mit der Testautomatisierung ist auf jeden Fall anzustreben.

Oftmals haben involvierte Personen zu hohe und unrealistische Erwartungen an die Testautomatisierung. Wenn diese Erwartungen dann nicht innerhalb eines Zeitfensters befriedigt werden, wird die Testautomatisierung oftmals vernachlässigt oder gar gänzlich verlassen. Die unrealistischen Erwartungen entstehen oft durch Unkenntnis der verwendeten Tools und generelle Unkenntnis über die Möglichkeiten und Einschränkungen von Testautomatisierungen und werden bei Nichterfüllung oftmals als Verschwendung angesehen, auch wenn die Automatisierung realistisch gesehen einen hohen Wert für die Organisation hat.

Die Einführung von Testautomatisierung, wie auch der Ankauf von Automationswerkzeugen von Drittanbietern, ist mit hohen Kosten verbunden. Der Return On Investment (ROI) tritt erst nach vielen Testzyklen und Testfällen auf. Das ist oftmals bei kleineren Projekten eine Hürde, eine Testautomatisierung zu implementieren. Gleichzeitig ist der Testprozess der erste, der bei Zeitknappheit und sich nähernden Deadlines vernachlässigt oder ignoriert wird. Wenn der Test nicht als wertvolle Aktivität des Geschäfts empfunden wird, wird auch eine Testautomatisierung nicht als wertvoll angesehen. Jedoch würde man bei hoher Qualität und geringer Fehlerwahrscheinlichkeit eher langfristige Erfolge und wirtschaftliche Partnerschaften erzielen und man könnte bei Nachfolgeprojekten die bereits erstellten und gespeicherten Testfälle verwenden.

Durch die Komplexität der Automatisierungstools müssen bestehende Mitarbeiter ein neues System erlernen. Bei Unit-Tests wären die Programmierer bestimmt, diese zu implementieren. Oftmals sind in Unternehmen zu wenig Mitarbeiter vorhanden oder zu wenig Zeit, um ein neues System zu erlernen. Als Work-Around werden dann manuelle Tests durchgeführt, da diese einzeln gesehen schneller zu erledigen sind als Testfälle zu erstellen, diese zu prüfen und dann auch auszuführen.

Die Expertise für die Automatisierungstools muss entweder durch Trainings, Tutorials oder Selbstschulung erworben werden. Durch die oben beschriebene Komplexität verursachen diese Trainings enorme Kosten und Zeit, die in vielen Fällen nicht vorhanden ist. Eine Anwendung mit fehlender Expertise würde in ineffizienter Automatisierung, ständigen Anpassungen und steigender Enttäuschung sichtbar werden.

Eine weitere Hürde könnten Einschränkungen der betroffenen Systeme sein, so könnte das Automatisierungstool Limitierungen enthalten oder das zu testende System ist für einen komplette Abdeckung durch Testautomatisierung nicht zugänglich. Weiters ist darauf zu achten, dass die Automatisierungsplattform selbst ein stabiles System ist. Work-Arounds in der Automatisierungsumgebung, nur um Testfälle durchlaufen zu können, führen zu einer ineffizienten Verwendung des Systems und müssten

gegebenenfalls bei jedem Software-Update neu untersucht werden. Gleichzeitig muss aber das zu testende System und dessen Kommunikation und Interaktion mit dem Automatisierungstool überprüft werden. Das beste Werkzeug hat keinen Nutzen, wenn das zu testende System die Kommunikation nicht zulässt.

2.5.5 Best Practices / Bewährte Vorgehensweisen

Auch heute noch haben die Prinzipien von Polo aus dem Jahre 2013 Gewicht in der Einführung und dem Betrieb von Testautomatisierungen. Da diese Testautomatisierung aber abhängig von Open-Source-Communitys oder kommerziellen Herstellern ist fehlen hier vielfach Normen und Standards. Lediglich das TTCN-3 (Testing and Test Control Notation) von ETSI (European Telecommunication Standards Institute) bildet hier eine Ausnahme, diese sind aber auf spezifische Bereiche, wie Telekommunikation oder Automobil-Sektor, beschränkt. Durch die normfreie Technologie wird es derzeit und auch in naher Zukunft nicht einfach möglich sein, Testfälle und -ergebnisse von einem Testwerkzeug in ein anderes zu übertragen. Weiters werden auch die Entwickler und Tester für bestimmte Werkzeuge ausgebildet, bei Wechsel auf ein anderes System würden hier wiederholte Ausbildungskosten anfallen (Bucsics et al. 2015).

Dennoch haben sich aus den Prinzipien folgende bewährte Vorgehensweisen durch empirische Methoden durchgesetzt (Wang et al. 2022):

- Definition einer effektiven Testautomatisierungsstrategie
  Diese Strategie legt fest, welche Bereiche automatisiert werden, welche weiterhin manuell abgearbeitet werden und welche automatisierten Tests wann durchgelaufen werden sollen. Weiters wird bestimmt, welche Entwickler welche Modul-Tests erstellen und wer für die Durchführung der Testautomatisierung zuständig ist.
- Genügend Ressourcen bereitstellen
  Für das Erstellen, Warten und Durchführen der Automatisierung werden bei den Entwicklern mehr Ressourcen benötigt. Durch ungenügende Ressourcen bei der Umstellung Automatisierung und Instandhaltung dieser werden weiterhin viele Ressourcen durch manuelle Tests gebunden.
- Kompetente Testersteller und -ausführer
  Durch Kompetenzen bei der Erstellung und Ausführung der automatisierten Tests können die Ergebnisse in kurzer Zeit eingeordnet und Testfälle bei Bedarf angepasst werden. Durch fehlendes Wissen und Skills könnten Testfälle ausfallen, falsche Ergebnisse liefern oder die Ergebnisse würden zu spät interpretiert und ausgewertet werden.
- Auswahl der richtigen Testwerkzeuge
  Wie in 2.5.3 Entscheidungsgrundlagen zur Testautomatisierung beschrieben muss die Auswahl der Testwerkzeuge gewissenhaft erfolgen. Die Werkzeuge

müssen auf das zu testende System zugeschnitten sein um eine höchstmögliche Testautomatisierung bewerkstelligen zu können.

- Bereitstellen einer stabilen und aktuellen Testumgebung
  Die Testumgebung muss von der Testautomatisierung aufrufbar sein und muss jenen Software-Stand beinhalten, der auch auf das Produktiv-System ausgeliefert werden soll. Wäre die Testumgebung auf einem nicht-aktuellen Software-Stand könnten Testfälle fälschlicherweise fehlschlagen oder neu implementierte Funktionalitäten nicht überprüft werden.
- Software konzipieren für automatisierte Tests
  Die zu überprüfende Software soll so konzipiert sein, dass Tests einfach erstellt werden können. Dies bezieht sich auf den Modul-Test, bei dem mittels Programmcode die Funktionalität überprüft werden kann. Ein bereits erwähnter Ansatz dazu ist unter 2.5.2 Test Driven Development (TDD) zu finden.
- Bereitstellung hochqualitativer Testdaten
  Die Testdaten müssen in entsprechender Anzahl vorhanden sein. Sie müssen unterscheidbar sein und auch dokumentiert werden, ob und wieso diese Daten erfolgreich sind oder fehlschlagen. Die für die Testfälle verwendeten Daten sind im optimalen Fall aus erfolgreichen Werten, falschen Eingaben und Grenzwerten.
- Entwicklung hochqualitativer Testfälle und Testskripte
  Testfälle sollen so kurz wie möglich, und so lange wie nötig konzipiert sein. Diese Testfälle sollen dabei nur eine Funktionalität abdecken und keinen ganzen Bereich. Dafür bieten sich Test-Suites an, die mehrere Testfälle beinhalten. Weiters sollen die Testfälle auch leicht verständlich und anpassbar sein, um auf geänderte Funktionalitäten oder Updates schnell reagieren zu können.
- Automatisierung des Testorakels
  Das Testorakel ist die Quelle zur Überprüfung des vorausgesagten Ergebnisses, verglichen mit dem tatsächlichen Ergebnis des Testfalls. Dieses Orakel kann ein Benutzerhandbuch, das Wissen einer Person oder ein Benchmark sein. Durch die Automatisierung des Testorakels werden Ressourcen geschont, da die Überprüfung des Ergebnisses nicht mehr durch eine Person vorgenommen wird. Zu beachten ist, dass auch das Testorakel immer auf dem neuesten Stand sein muss, da bei geänderten Funktionalitäten sonst falsche Ergebnisse geliefert werden könnten.
- Effiziente und effektive Analyse der Ergebnisse
  Nach dem Testdurchlauf wird durch eine zentrale Ergebnisspeicherung gewährleistet, dass auf einen Blick alle relevanten Resultate ersichtlich sind. So kann zeitnahe auf Fehler oder Anomalien reagiert werden oder die Software bei einem erfolgreichen Durchlauf aller Testfälle für das Produktiv-System freigegeben werden. Durch eine Implementierung eines Zugriffskontrollsystems können nur befugte Personen die Ergebnisse analysieren und die weitere

Vorgehensweise bestimmen. So sind auch unbefugte Personen nicht berechtigt über die Freigabe eines Software-Stands auf das Produktiv-Systems zu entscheiden.

- Einbindung neuer Technologien
  Die Testautomatisierung ist ein lebendes System. In regelmäßigen Abständen werden neue Funktionalitäten der Testwerkzeuge veröffentlicht oder neue „Best Practices“ angekündigt. Durch eine permanente Instandhaltung und Aktualisierung der Werkzeuge kann eine Veralterung dieser unterbunden werden. Eine neue Werkzeug-Version kann möglicherweise Funktionalitäten beinhalten, die neue Bereiche für die Testautomatisierung öffnet oder liefert verbesserte Ergebnisse für die einzelnen Testfälle.

2.5.6 Testautomatisierung mit Artificial Intelligence/Machine Learning

Ein Bereich, der in den letzten Jahren in der Software-Entwicklung einen enormen Aufschwung erlebt hat ist künstliche Intelligenz (KI / AI – Artificial Intelligence) und Machine Learning (ML).

AI/ML kann auch bei der Testautomatisierung eine sinnvolle Unterstützung sein. So könnte bei Entwicklung einer neuen Funktionalität in einem Modul automatisch ein zugehöriger Modultest erstellt werden, der die Funktionalität dieser neuen Implementierung überprüft. Weiters könnte AI/ML aufgrund der Steuerelemente, wie OK-Button oder Abbrechen, ein geeignetes Testkonzept, geeignete Testfälle und qualitativ hochwertige Daten erstellen, die zu diesen Bedienelementen passen und zusätzlich auch die Positionierung bei verschiedenen Auflösungen oder Plattformen überprüfen.

Bei der Erstellung der Testdaten könnte AI/ML bei den eingegebenen Daten im Produktiv-System mitlernen und so oft verwendete Werte-Bereiche speichern und diese als Testdaten bereitstellen oder gleich verwenden.

„Selbstheilungs-Feature“ von AI/ML bezieht sich auf die Instandhaltung und Wartung von Testfällen. Sollte sich im Code etwas ändern oder erfährt die Version ein Update kann AI/ML selbst die betroffenen Testfälle ermitteln und diese auf die neueste Version bringen oder umbauen, womit Entwickler-Ressourcen geschont werden würden und die Effizienz der erstellten Testfälle zunimmt. (Pham et al. 2022)

Weiters könnte in Zukunft AI/ML bereits bei der Entwicklung von neuen Funktionalitäten unterstützen und aufgrund ermittelter Daten den Entwickler auf Überläufe oder instabile Rechenoperationen hinweisen. So würden Software-Tests durch AI/ML parallel zur Implementierung ablaufen und die Testphase weiter verkürzen.

Folgend ist eine Auflistung einer Auswahl von Unternehmen, die Testautomatisierung in Verbindung mit AI/ML anbieten:

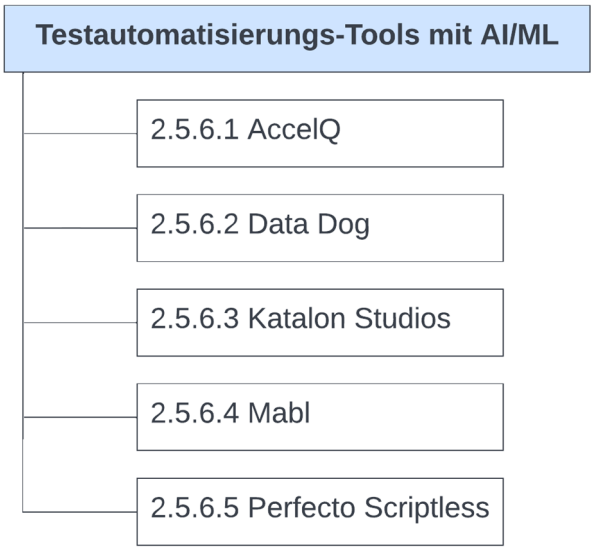

Abbildung 8: Auswahl Testautomatisierung AI/ML (eigene Abbildung, 2023)

2.5.6.1 AccelQ

AccelQ ist ein cloud-basiertes Werkzeug zur code-freien Erstellung von Testfällen. Der Fokus liegt auf der Automatisierung von GUIs, APIs, Desktop- und mobilen Apps. AI/ML wird zur Erzeugung von Testfällen eingesetzt und zusätzlich können Benutzer mit natursprachlichen Eingaben Testfälle ohne Programmierkenntnissen anlegen lassen.

2.5.6.2 Data Dog

Data Dog analysiert die Infrastruktur und die Performance der Software mittels Machine Learning und gibt den Entwicklern bei Spitzen Rückmeldung, ohne manuelle Einrichtung der Alarmsituationen. So können Fehlersituationen und Performance-Einbrüche in Datenbanken, Datenbankabfragen und Netzwerkprobleme erörtert werden.

2.5.6.3 Katalon Studios

Dieses Ende-zu-Ende-Test-Werkzeug ist spezialisiert auf AI-unterstützte visuelle Tests und verfügt zusätzlich über eine Selbstheilungs-Funktion, um fehlende Elemente in Websites durch passende zu ersetzen.

2.5.6.4 Mabl

Auch Mabl ist ein Ende-zu-Ende-Test-Werkzeug, das Testfälle bei Updates automatisch anpassen kann, um jederzeit aktuelle Testfälle anbieten zu können.

2.5.6.5 Perfecto Scriptless

Auch dieses code-freie Werkzeug verfügt über eine ML-Unterstützung, die gleichartige Muster in verschiedenen Tests erkennt und Selbstheilung durchführt.

3. KONZEPTIONELLER VORGEHENS- UND LÖSUNGSANSATZ

3.1 Kapitelübersicht

In diesem Kapitel erkläre ich, welche Testfälle für die Hypothese „Durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka wird der zeitliche Aufwand um den Faktor 5 verringert“ automatisiert wurden und welche Zeitersparnis, oder auch Zeitverlust damit erreicht wurde. Vordergründig bezieht sich die Zeitersparnis oder -verlust auf die KeyUser der Fachabteilungen, jedoch werde ich für die Überprüfung der Hypothese auch die zusätzlich benötigten Ressourcen der Entwicklerseite beleuchten, die bei Einführung der Testautomatisierung und Erstellung der Test-Skripte doch enorm beansprucht werden.

3.2 Methodenübersicht

Um die für diese Arbeit erheblichen Testfälle aufzunehmen werden die KeyUser bei der Überprüfung der Funktionalität aufgenommen und mittels „Record and Playback“ Aufnahmenaufzeichnung gespeichert. So können diese Testfälle später automatisch mit dem unter 2.4.10 erwähnten RSAT durchgelaufen werden. Die Aufzeichnung der KeyUser liefert auch die benötigte Dauer, in der ein Testdurchlauf händisch durchgeführt wird und kann später durch Vergleiche mit dem automatisch durchlaufenen Test herangezogen werden. Weiters werden Modultests erstellt, die bei Implementierung neuer Funktionalität mittels unter 2.4.1 erwähnten MSTestV2 im Code angelegt werden. Auch hier können die Dauer manueller Tests und Modultests verglichen werden. Als empirische Untersuchung dient eine Primärdatenerhebung, da derzeit noch keine Daten und Zeiten über händisch durchgeführte Tests vorliegen.

3.2.1 Primärdatenerhebung

Wie im vorigen Absatz erwähnt werden die händisch durchgeführten Tests aufgezeichnet und deren Dauer so ermittelt. Für diese Arbeit wurden sechs Testfälle ermittelt, die quer über die Fachabteilungen verstreut sind und unterschiedlich automatisiert werden. So wird beispielsweise das Anlegen eines neuen Artikels mit dem Werkzeug RSAT realisiert, während das Umrechnen von Packungen auf Verkaufseinheiten mittels Modultest realisiert wird.

Abbildung 9 zeigt die Aufteilung der überprüften Testfälle und der Zugehörigkeiten in die verschiedenen Fachabteilungen:

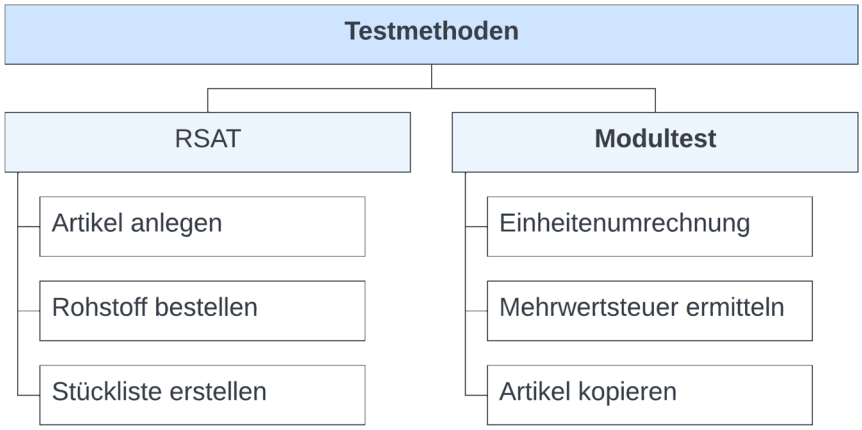

Abbildung 9: Kategorisierung der Testfälle (eigene Abbildung, 2023)

Artikel anlegen: Hier wird ein neuer Verkaufsartikel mit Nummer, Bezeichnung, Inhaltsstoffe und Klassifizierung (wie Kühltemperatur und Mindesthaltbarkeitsdatum) angelegt. Diese Artikelanlage erfolgt gänzlich über das Frontend von Microsoft Dynamics 365 for Finance & Operations. Der Test überprüft die Gültigkeit und das Vorhandensein von Werten und Pflichtfeldern.

Rohstoff bestellen: Bei der Bestellung von Rohstoffen werden die eingegebene Menge, das Bestelldatum, sowie das Lieferdatum überprüft und ob die Bestell-E-Mail an den Lieferanten geschickt wurde.

Stückliste erstellen: Bei der Erstellung der Stücklisten wird überprüft, ob die eingegebenen Artikelnummern und Rohstoffnummern existieren und ob die Mengen einen sinnvollen Wert enthalten.

Einheitenumrechnung: Eine Verkaufseinheit besteht beispielsweise aus 8 Frischteigen, mittels Modultest wird hier überprüft, ob auf den richtigen Umrechnungsfaktor zugegriffen wird.

Mehrwertsteuer ermitteln: Bei der Journalbuchung ist abhängig von Land und der mehrwertsteuer-freien Intercompany-Lieferung ein anderer Satz der Mehrwertsteuer heranzuziehen. Der Modultest überprüft die Korrektheit dieses Satzes.

Artikel kopieren: Bei der Kopie eines Artikels werden seine Attribute auf einen neuen, leeren Artikel übertragen. Durch Überprüfung der einzelnen Attribute wird überprüft, ob der Artikel komplett übertragen wurde oder ob ein Feld leer geblieben ist beziehungsweise falsch übermittelt wurde.

3.2.2 Proof-of-Concept

Ausgehend von der ermittelten Dauer der händisch durchgeführten Testfälle und der gespeicherten Aufzeichnungen werden diese danach miteinander verglichen, wobei jeweils pro Testfall einmal mit korrekten und einmal mit falschen Testdaten verglichen wird. Schlussendlich wird zu den automatischen Tests die Entwicklungsdauer und gegebenenfalls die Instandhaltungsdauer einzelner Testfälle ermittelt und addiert.

Da durch die Installation und der Erstellung der Testfälle anfänglich viele Ressourcen der Software-Entwicklungsabteilung gebunden werden, werde ich auch erörtern, wie viel Zeit durch den Einsatz von stabiler und bereits erstellten Testfälle eingespart wird.

Das Proof-of-Concept wird auf jeden Fall in Zukunft die manuellen Tests bei Wewalka ergänzen oder sogar ablösen. Auch wenn die Hypothese nicht vollständig bestätigt wird dient die Testautomatisierung doch der Entlastung der KeyUser und kann außerhalb der gewöhnlichen Arbeitszeiten wiederholt eingesetzt werden.

4. ANWENDUNG DES LÖSUNGSVORSCHLAGS

4.1 Kapitelübersicht

In diesem Kapitel wird erklärt, wie eine Aufgabenaufzeichnung im ERP-System vorgenommen wird, die danach im RSAT-Tool in einen Testfall umgewandelt und wiederholt automatisch ausgeführt werden kann.

Um den Umfang einzuschränken wird nur die Erstellung der Aufzeichnung für den Testfall „Ermittlung der höchsten Artikelnummer“ beschrieben. Die übrigen Aufzeichnungen der verwendeten Testfälle des automatisierten Tests bei Wewalka beruhen aber auf der gleichen Funktionsweise wie der gezeigte Testfall.

Weiters wird die Test Suite „Artikelanlage“ mit den fünf Testfällen beschrieben, die in bestimmter Reihenfolge und immer gemeinsam auszuführen sind. Eine Test Suite ist eine Zusammenstellung von mehreren Testfällen für den Test einer Komponente des Systems.

Hier kann der Output eines Testfalls der Input eines folgenden Tests sein (ISTQB, 2015). In der Test Suite „Artikelanlage" ist beispielsweise die höchste Artikelnummer der Output des Testfalls „Ermittlung der höchsten Artikelnummer" der Input der folgenden Testfälle „Artikel anlegen“, „Artikel löschen“ und „Produkt löschen“. Diese Testfälle benötigen die zuvor ermittelte Artikelnummer, um einen neuen Artikel anlegen und den zuvor erstellten wieder löschen zu können.

Folgend sind die erstellten Unit-Tests, die direkt im Quellcode programmiert sind und automatisch bei jedem Build-Vorgang abgespielt werden.

4.2 Erstellung einer Aufgabenaufzeichnung

In diesem Beispiel wird das Formular „Freigegebene Produkte“ aufgerufen und durch Filterung und Sortierung der Spalten die höchste vergebene Artikelnummer ermittelt. Diese wird gespeichert und bei den folgenden Testfällen um 1 erhöht, um eine freie Artikelnummer zu bekommen und zu verwenden.

Durch Klick auf das Einstellungen-Symbol in der Titelleiste von Microsoft D365 Finance & Operations und Auswählen von „Aufgabenaufzeichnung“ kann nach Eingabe des Namens und der Beschreibung die Aufzeichnung gestartet werden.

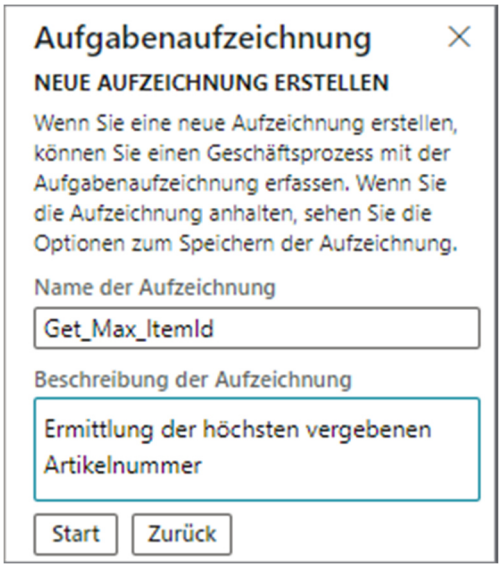

Abbildung 10: Erstellen einer Aufzeichnung (eigene Abbildung, 2023)

Nun wird wie gewünscht durch das System navigiert, die Schritte werden automatisch aufgezeichnet und mitprotokolliert, wie im System auf der rechten Seite ersichtlich.

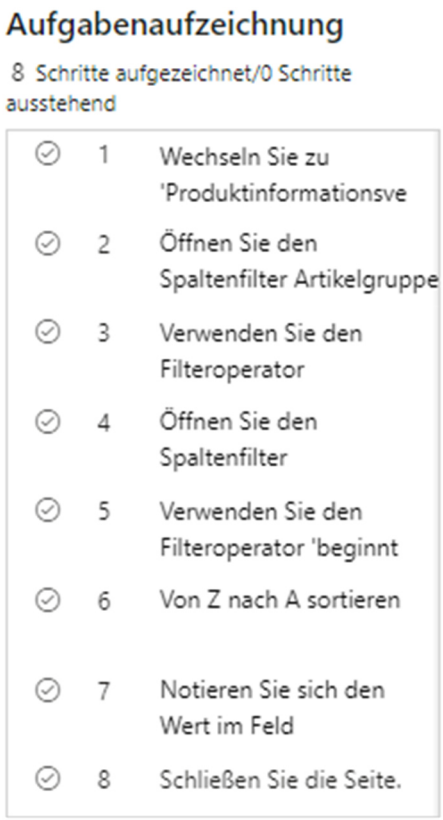

Abbildung 11: Fertige Aufgabenaufzeichnung (eigene Abbildung, 2023)

Die verwendeten Filter, die Sortierung und das Speichern benötigter Variablen für diesen Testfall werden in der folgenden Passage erläutert.

Ist der Testfall abgeschlossen wird dieser gespeichert und im RSAT-Tool zu einem Testfall mit Excel-Parameter-File umgewandelt.

4.3 Test Suite „Artikelanlage“

Der Testfall „Artikel anlegen“ besteht aus folgenden fünf Testfällen und wird der Übersicht halber in der Test Suite „Artikelanlage“ zusammengefasst:

1) Ermittlung der höchsten Artikelnummer
2) Artikel anlegen
3) Artikel prüfen
4) Artikel löschen
5) Produkt löschen

Während im Gegensatz zu den Unit Tests hier mit der Oberfläche gearbeitet wird und echte Prozesse abgearbeitet werden, sind angelegte Datensätze über die Dauer der Testfälle hinaus im System vorhanden. Um die Datenmenge gering zu halten wird der angelegte Artikel nach Überprüfung wieder aus dem System gelöscht. Eine Eigenheit der Firma Wewalka ist bei Anlage eines Artikels wird dieser auch als Produkt erzeugt. Dieses Produkt muss explizit gelöscht werden, um den angelegten Artikel wirklich restlos aus dem System zu entfernen.

4.3.1 Ermittlung der höchsten Artikelnummer

Um ein neues Teigprodukt dynamisch anlegen zu können wird vor der Anlage die höchste vergebene Artikelnummer mittels einer Aufgabenaufzeichnung ermittelt, die zuvor mit dem RSAT-Tool erstellt werden muss.

Eine Eigenheit der Firma Wewalka, die hierbei bedacht werden muss ist die Nummerierung der verschiedenen Produktgruppen. Teige sind als Halbfertigfabrikate (HF) gekennzeichnet, die das Prefix „160“ besitzen. Weiters existieren Teigrezepte, die als Halbfertigfabrikate mit Prefix „15“ gekennzeichnet sind. Jedoch sind Folien - als Verpackungsmaterial - gekennzeichnet und beginnen mit dem Prefix „160-“. Somit reicht eine alleinige Filterung „beginnt mit 16“ nicht aus.

Zwei mögliche Herangehensweisen haben sich als hilfreich erwiesen:

- Filter auf Artikelnummer „entspricht“ mit Wert „160????“
  Bei der Filterung mit dem Parameter „entspricht“ können Wildcards eingegeben werden. So entspricht „160????“ einem Wert, der mit „160“ beginnt und sieben Stellen lang ist.

Sollten in ferner Zukunft die Teigproduktenummern achtstellig werden, würde dieser automatische Test aber fehlschlagen. Deswegen habe ich mich für die zweite Herangehensweise entschieden.

- Filter auf Artikelnummer „beginnt mit“ mit dem Wert „160“ und Filterung der Artikelgruppe „entspricht genau“ HF

Somit werden nur Halbfertigfabrikate mit Prefix „160“ ausgewählt, also die Teigprodukte. Folien und Rezepte werden hier ausgefiltert.

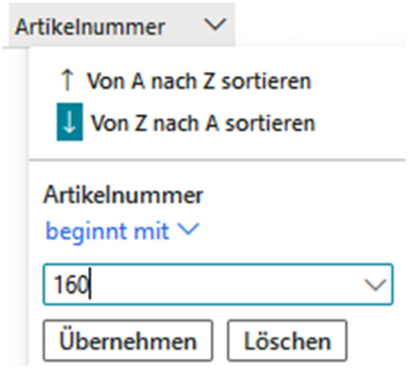

Abbildung 12: Filter von Artikelnummer (eigene Abbildung, 2023)

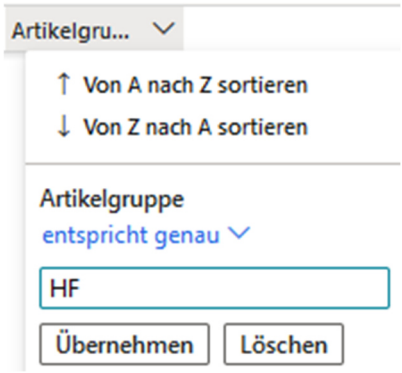

Abbildung 13: Filter von Artikelgruppe (eigene Abbildung, 2023)

Die zwei Filter der Abbildungen 12 und 13 beziehen sich auf die Schritte 2 bis 5 der Abbildung 11. Schritt 6 „Von A nach Z sortieren“ ist auf Abbildung 12 ersichtlich.

Nach erfolgreicher Filterung und Sortierung wird der oberste Wert als Variable in der Aufgabenaufzeichnung gespeichert. Die Aufzeichnung der „Ermittlung der höchsten Artikelnummer“ ist nun fertig und kann im RSAT-Tool zu einem Testfall mit Excel-Parameter-File und den Testschritten umgewandelt werden.

Excel-File mit Parametern und Testschritten

Field	Value
Test Case ID	1232
Recording Name	Ermittlung der höchsten Artikelnummer
Playback Order	0
Company	SOL
Test User	
Abort test suite execution on failure	WAHR
Fail on warning message in the Infolog	FALSCH

Saved variables
{{ItemId_1232}}

Tabelle 1: Parameter des Testfalls

Im General Tab des Excel Files werden globale Parameter für den Testdurchlauf gesetzt und können bei Bedarf angepasst werden.

„Test Case ID“ zeigt die automatisch vergebene ID, durch die der Testfall identifiziert und in Azure Dev Ops gefunden werden kann.

„Recording Name“ zeigt den Namen der Aufgabenaufzeichnung. Mittels „Company“ kann hier zwischen Mandanten des Unternehmens gewechselt werden. So gibt es neben dem Wewalka-Werk in Sollenau (SOL) eine Produktionsstätte in Celldömölk in Ungarn (CDM), deren Masken, Formulare und Prozesse sich gelegentlich unterscheiden und an Sollenau angepasste Testfälle womöglich in Celldömölk fehlschlagen könnten. Da der automatisierte Test erst in Sollenau eingeführt wird ist dieser Wert bei allen Testfällen mit SOL gefüllt.

Beim Parameter „Test User“ kann der Testfall aus Sicht und Berechtigung eines oder mehrerer spezifischer User durchgeführt werden. So können in Zukunft Berechtigungsthemen oder Zugriffe auf Menüs und Formulare überprüft werden, die nur bestimmten Personen oder Personengruppen zugänglich sind. Bei leerbleibendem Feld wird der aktuelle Benutzer für den Testfall herangezogen.

Der Parameter „Abort test suite execution on failure” bewirkt ein Abbrechen der Test Suite, wenn ein Testfall darin fehlschlägt. Dies ist hilfreich, um folgende Testfälle durch einen fehlschlagenden Test nicht fälschlicherweise zu manipulieren.

Der letzte Parameter „Fail on warning messag in the Infolog” lässt einen Testfall fehlschlagen, wenn das Informations-System des ERP-Systems eine Warnung ausgibt. Ein Beispiel dieser Warnungen ist in der kommenden Abbildung 14 zu sehen.

Bei den „Saved variables" sind die Variablen gespeichert, die bei der Aufgabenaufzeichnung markiert wurden, im Falle der Ermittlung der höchsten vergebenen Artikelnummer wurde diese als einzige Variable gespeichert.

Unter dem Reiter „MessageValidation“ können Nachrichten des Informations-Systems des ERP-Systems überprüft werden. Wird beispielsweise versucht ein Produkt ohne Artikelnummer anzulegen, wird vom System selbst eine Warnung ausgegeben. Mit der MessageValidation kann überprüft werden, ob die Fehlermeldung tatsächlich erscheint und ob die richtige Fehlermeldung ausgegeben wird. Das System liefert bei Eingabe eines Produktes ohne Artikelnummer den Hinweis „Feld 'Artikelnummer' muss ausgefüllt werden.“, wie Abbildung 14 zeigt.

Abbildung 14: Nachricht/Warnung von Microsofts D365 Finance & Operations (eigene Abbildung, 2023)

Im Excel-Tab TestCaseSteps sind die Schritte zu sehen, die bei der Erstellung des Testfalls getätigt würden. Bei „Erste freie Artikelnummer ermitteln“ werden die acht Testschritte angegeben:

Step	Action	Field	Value	Pause
1	*Navigate to: EcoResProductDetailsExtended ("ecoresproductdetailsextendedgrid")*			
2	*Verwenden Sie den Filteroperator 'entspricht genau', um den Filterwert 'HF' im Feld 'Artikelgruppe' einzugeben.*	**WEWItemGroupId: Is**	HF	
3	*Verwenden Sie den Filteroperator 'beginnt mit', um den Filterwert '160' im Feld 'Artikelnummer' einzugeben.*	**ItemId: BeginsWith**	160	
4	*Von Z nach A sortieren*			
5	*Klicken Sie in der Liste auf den Link in der ausgewählten Zeile.*			
6	*Klicken Sie im Aktivitätsbereich auf Optionen.*			
7	*Notieren Sie sich den Wert im Feld Artikelnummer für Ihre Unterlagen {{ ItemId_1232 }}*			
8	*Schließen Sie die Seite.*			

Tabelle 2: Testschritte „Artikel prüfen“

Die Navigation durch die Menüs und die überprüften, sowie eingegebenen Werte können hier nochmal eingesehen und auch verändert werden.

So wird beschrieben, dass die Seite "Freigegebene Produkte" (Name im Code: EcoResProductDetailsExtended) aufgerufen wird und danach die Filterung und die Sortierung stattfindet.

Bei Schritt 7 wird die zuvor gespeicherte Variable erwähnt. In dieser ist die höchste vorhandene Artikelnummer vorhanden, die im nächsten Testfall "Artikel anlegen" benötigt wird. Leider ist die deutsche Übersetzung des Schrittes etwas unglücklich gewählt, die Variable wird hier automatisch gespeichert, man muss diese nicht selbst für die Unterlagen notieren.

In der Spalte „Field“ wird die Spalte des ERP-Systems angegeben, die gefiltert werden soll mit der Filteroption. So wird bei Schritt 2 „Artikelgruppe entspricht genau HF“ und bei Schritt 3 „Artikelnummer beginnt mit 160“ angewendet.

In der Spalte „Pause“ können Wartezeiten zwischen den einzelnen Schritten angegeben werden. Eine hilfreiche Funktion, wenn man bei größeren Datenmengen auf das Laden des Formulars warten muss.

Im Excel-Tab CustomParameters können dynamische Parameter eingegeben werden, die für den Testfall relevant sind. Bei der Ermittlung der höchsten Artikelnummer sind keine Parameter nötig. Ein Beispiel für CustomParameters liefert der nächste Testfall "Artikel anlegen".

4.3.2 Artikel anlegen

Im zweiten Testfall der Test Suite „Artikelanlage“ wird nun ein Artikel/Teigprodukt mit benötigten Werten angelegt. Auch dies geschieht erstmalig über eine Aufgabenaufzeichnung, bei der aber mehrere Felder als Variablen gespeichert werden müssen. Nach der Umwandlung in einen ausführbaren Testfall erhält man ein Excel-File mit den gespeicherten Variablen:

Saved variables
{{ProductNumber_1229}}
{{Name_1229}}
{{WEWAPositioning_1229}}
{{Margarine_WEW_1229}}
{{WEWActionGroupId_1229}}
{{WEWProductGroup_1229}}
{{WEWDoughForm_1229}}
{{WEWDoughClassification_1229}}
{{WEWNetDepth_1229}}
{{WEWNetHeight_1229}}
{{WEWNetWidth_1229}}
{{WEWItemWeightPerRawPastry_1229}}
{{WEWRecommendedBBDFromRamp_1229}}
{{WEWUniqueSellingProposition_1229}}
{{WEWClassificationPalmFree_1229}}

Tabelle 3: Gespeicherte Variablen des Testfalls

Die Werte der Pflichtfelder werden in separate Variablen gespeichert, damit sie bei der folgenden Prüfung des neu angelegten Artikels auf korrekte Eintragung überprüft werden können. Die genaue Erklärung der einzelnen Felder folgt bei den Test Case Steps.

Bei „Artikel anlegen" sind 55 Schritte für die Abarbeitung notwendig, um einen neuen Artikel korrekt anzulegen. Ich habe hier aus Gründen der Übersicht die Navigationsschritte ausgeblendet und werde nur die wichtigsten Schritte erklären:

Step	Action	Field	Value
1	*Navigate to: EcoResProductDetailsExtended ("ecoresproductdetailsextendedgrid")*		
…			
4	*Geben Sie im Feld Produktnummer einen Wert ein.*	**Produkt-nummer**	=VALUE({{{ ItemId_1232 }}})+1
5	*Notieren Sie den Wert im Feld Produktnummer {{ProductNumber_1229}}*		
6	*Geben Sie im Feld Produktname einen Wert ein.*	**Produktname**	BLT Inline 275g palmfrei
7	*Notieren Sie sich den Wert im Feld Produktname {{Name_1229}}*		
…			
25	*Geben Sie im Feld 'Positionierung' einen Wert ein, oder wählen Sie einen Wert aus.*	**Positionierung**	STA
26	*Notieren Sie sich den Wert im Feld Positionierung {{WEWAPositioning_1229}}*		
27	*Wählen Sie im Feld 'Margarine' die Option 'Ja' aus.*	**Margarine**	WAHR
28	*Notieren Sie sich den Wert im Feld Margarine {{Margarine_WEW_1229}}*		
29	*Wählen Sie Im Feld Ereignisgruppe eine Option aus.*	**Ereignisgruppe**	3
30	*Notieren Sie sich den Wert im Feld Ereignisgruppe {{WEWActionGroupId_1229}}*		
31	*Wählen Sie Im Feld Verwendungsgruppe eine Option aus.*	**Verwendungs-gruppe**	1
32	*Notieren Sie sich den Wert im Feld Verwendungsgruppe {{WEWProductGroup_1229}}*		
33	*Wählen Sie im Feld Form eine Option aus.*	**Form**	2
34	*Notieren Sie sich den Wert im Feld Form {{WEWDoughForm_1229}}*		
35	*Wähen Sie Im Feld Teigklassifizierung eine Option aus.*	**Teig-klassifizierung**	3
36	*Notieren Sie sich den Wert im Feld Teigklassifizierung {{WEWDoughClassification_1229}}*		

Step	Action	Field	Value
37	*Geben Sie im Feld 'Netto-Tiefe' eine Zahl ein.*	**Netto-Tiefe**	400
38	*Notieren Sie sich den Wert im Feld Netto-Tiefe {{WEWNetDepth_1229}}*		
39	*Geben Sie im Feld 'Netto-Höhe' eine Zahl ein.*	**Netto-Höhe**	2,9
40	*Notieren Sie sich den Wert im Feld Netto-Höhe {{WEWNetHeight_1229}}*		
41	*Geben Sie im Feld 'Netto-Breite' eine Zahl ein.*	**Netto-Breite**	240
42	*Notieren Sie sich den Wert im Feld Netto-Breite {{WEWNetWidth_1229}}*		
43	*Geben Sie im Feld 'Gewicht Teigling' eine Zahl ein.*	**Gewicht Teigling**	275
44	*Notieren Sie sich den Wert im Feld Gewicht Teigling {{WEWItemWeightPerRawPastry_1229}}*		
…			
46	*Geben Sie im Feld 'Empfohlenes MHD ab Rampe' eine Zahl ein.*	**Empfohlenes MHD ab Rampe**	35
47	*Notieren Sie sich den Wert im Feld Empfohlenes MHD ab Rampe {{WEWRecommendedBBDFromRamp_1229}}*		
48	*Geben Sie im Feld USP/Auslobung einen Wert ein.*	**USP/ Auslobung**	goldgelb, schöne Schichtenbildung
49	*Notieren Sie sich den Wert im Feld USP/Auslobung {{WEWUniqueSellingProposition_1229}}*		
50	*Wählen Sie im Feld 'Palmfrei' die Option 'Ja' aus.*	**Palmfrei**	WAHR
51	*Notieren Sie sich den Wert im Feld Palmfrei {{WEWClassificationPalmFree_1229}}*		
…			

Tabelle 4: Testschritte „Artikel anlegen"

Bei Schritt 4 wird die zuvor ermittelte Artikelnummer mittels Excel-Funktion um 1 erhöht und der Produktnummer zugewiesen. (Im Wewalka-ERP wird bei Eintragung der Produktnummer automatisch die Artikelnummer gefüllt und umgekehrt).

Der Feldwert in den Testfallschritten ist wie folgt einzutragen, wobei hier Excel-Funktionen zum Einsatz kommen:

`'=VALUE({{ItemId_1232}})+1`

Zuerst wird der Wert der Variable ItemId_1232 in eine Zahl umgewandelt, danach wird diese um 1 erhöht. Diese Zahl wird nun als erste freie Artikelnummer genommen und der neue Artikel mit dieser Nummer angelegt. So hat man auf jeden Fall eine unverwendete Nummer, die zur Anlage eines neuen Artikels verwendet werden kann. Der Produktname wird im ausgeführten Testfall mit „BLT Inline 275g palmfrei“ festgelegt (BLT = Blätterteig).

Bei der Positionierung wird "STA" für Standard ausgewählt. Da dieses Produkt Margarine enthält und auch das Feld Margarine überprüft werden soll wird der Radio-Button aktiviert (auf JA gesetzt).

Bei der Ereignisgruppe zeigt Schritt 27 den Wert 3 an, dieser bedeutet im System "Frischteig", Verwendungsgruppe 1 bedeutet "Blätter", Form 2 ist ein eckiger Teig (1 = rund) und Teigklassifizierung 3 steht für "BLT InLine" – passend zu dem Produktnamen.

Die Maße des Teigprodukts werden auf Tiefe 400mm, Höhe 2,9mm und Breite 240mm festgelegt, zudem beträgt das Gewicht des Teiglings 275g.

35 Tage soll die empfohlene Mindesthaltbarkeitsdauer betragen, die Auslobung des Produkts wird mit „goldgelb, schöne Schichtenbildung" gefüllt und außerdem wird das Feld „Palmfrei“ auf JA gesetzt und später überprüft.

Wie vorhin erwähnt bietet sich hier eine Tabelle im Excel-Reiter Custom Parameters an, um eine übersichtliche Darstellung der verwendeten Parameter zu haben. Ein Beispiel für einen ähnlichen Testfall könnte wie folgt aussehen:

Produktnummer	=(VALUE({{EcoResProductDetailsExtended_InventTable_ItemId_1232_Copy}})+1)
Produktname	BLT Inline 375g palmfrei
Vorlage anwenden	Produkt CM - 16er HF (SOL)
Standardauftragstyp	Produktion
Standardlagerort	HAL
Standardlagerort	HAL
Standardlagerort	HAL
Positionierung	STA
Margarine	TRUE
Ereignisgruppe	Frischteige
Verwendungsgruppe	Blätter
Form	Eckig
Teigklassifizierung	BLT InLine
Netto-Tiefe	400
Netto-Höhe	2,9
Netto-Breite	240
Gewicht Teigling	375
Empfohlenes MHD ab Rampe	35
USP/Auslobung	goldgelb, schöne Schichtenbildung
Palmfrei	TRUE

Abbildung 15: Custom Parameters (eigene Abbilung, 2023)

Da es bei spezifischen Artikeln drei verschiedene Lagerorte als Standard geben kann, wird der Parameter dreimal in der Abbildung 15 angeführt. Für Anwender*innen des Testsystems und der Testautomatisierung sind diese drei Lager selbsterklärend, für unbefugte Benutzer*innen könnte aber hier Verwirrung entstehen.

Zu beachten ist, dass dann bei den Testfallschritten mittels richtiger Referenzierung auf die Werte zugegriffen wird, wie beispielsweise =‘Custom Parameters‘!B2

4.3.3 Artikel prüfen

Beim dritten Testfall werden die beim zweiten Testfall eingegebenen Werte auf korrekte Speicherung überprüft. Nach der Navigation zu der Artikelseite wird nach der neu erstellten Artikelnummer gefiltert, diese ausgewählt um zur Detail-Seite zu kommen und dort die Eintragungen überprüft, die beim vorigen Schritt in Variablen abgespeichert wurden. Zu sehen ist dies an der Spalte „Value“ in der folgenden Tabelle. Diese Variablen mit der Form {{Name_xxxx}} sind gleichlautend wie die in Tabelle 3: Gespeicherte Variablen des Testfalls „Artikel anlegen“ auf Seite 37.

Step	Action	Field	Value
1	*Navigate to: EcoResProductDetailsExtended ("ecoresproductdetailsextended grid")*		
2	*Verwenden Sie den Filter 'beginnt mit', um den Wert 'Artikelnummer' einzugeben.*	**ItemId: BeginsWith**	=VALUE({{ItemId_1232}}) + 1
3	*Klicken Sie in der Liste auf den Link in der ausgewählten Zeile.*		
4	*Klicken Sie im Aktivitätsbereich auf Optionen.*		
5	*Überprüfen Sie, ob der Wert für Artikelnummer dem Filterwert entspricht.*	**Validate Artikelnummer**	=VALUE({{ItemId_1232}}) + 1
6	*Überprüfen Sie, ob der Wert für Produktnummer dem Filterwert entspricht ist.*	**Validate Produktnummer**	=VALUE({{ItemId_1232}}) + 1
7	*Überprüfen Sie, ob der Wert für Produktname 'BLT Inline 275g palmfrei' ist.*	**Validate Produktname**	={{Name_1229}}
8	*Überprüfen Sie, ob der Wert für Positionierung 'STA' ist.*	**Validate Positionierung**	={{WEWAPositioning_1229 }}

Step	Action	Field	Value
9	*Überprüfen Sie, ob der Wert für Ereignisgruppe '3' ist.*	**Validate Ereignisgruppe**	={{WEWActionGroupId_1229}}
10	*Überprüfen Sie, ob der Wert für Margarine 'True' ist.*	**Validate Margarine**	={{Margarine_WEW_1229}}
11	*Überprüfen Sie, ob der Wert für Verwendungsgruppe '1' ist.*	**Validate Verwendungs-gruppe**	={{WEWProductGroup_1229}}
12	*Überprüfen Sie, ob der Wert für Form '2' ist.*	**Validate Form**	={{WEWDoughForm_1229}}
13	*Überprüfen Sie, ob der Wert für Teigklassifizierung '3' ist.*	**Validate Teigklassifizierung**	={{WEWDoughClassification_1229}}
14	*Überprüfen Sie, ob der Wert für Netto-Tiefe '400' ist.*	**Validate Netto-Tiefe**	={{WEWNetDepth_1229}}
15	*Überprüfen Sie, ob der Wert für Netto-Höhe '2.9' ist.*	**Validate Netto-Höhe**	={{WEWNetHeight_1229}}
16	*Überprüfen Sie, ob der Wert für Netto-Breite '240' ist.*	**Validate Netto-Breite**	={{WEWNetWidth_1229}}
17	*Überprüfen Sie, ob der Wert für Gewicht Teigling '275' ist.*	**Validate Gewicht Teigling**	={{WEWItemWeightPerRawPastry_1229}}
18	*Überprüfen Sie, ob der Wert für Empfohlenes MHD ab Rampe '35' ist.*	**Validate Empfohlenes MHD ab Rampe**	={{WEWRecommendedBBDFromRamp_1229}}
19	*Überprüfen Sie, ob der Wert für USP/Auslobung 'goldgelb, schöne Schichtenbildung' ist.*	**Validate USP/Auslobung**	={{WEWUniqueSellingProposition_1229}}
20	*Überprüfen Sie, ob der Wert für Palmfrei 'True' ist.*	**Validate Palmfrei**	={{WEWClassificationPalmFree_1229}}

Tabelle 5: Testschritte "Artikel prüfen"

Bei diesem Testfall werden keine neuen Variablen und Werte gespeichert, da hier nur die Prüfung der zuvor erstellten Variablen und Werte durchgeführt wird. Durch diese Überprüfung wird festgestellt, ob die bei Testfall „Artikel anlegen“ eingegebenen Werte korrekt erfasst und den richtigen Feldern zugeordnet wurde. Weiters erfolgt dadurch eine Überprüfung, ob die Felder durch eine neue Entwicklung noch verfügbar und ansteuerbar sind und ob möglicherweise ein Update seitens Microsoft die überprüften Felder ausgeblendet oder entfernt hat. Sollte dieser Testfall korrekt durchgelaufen sein wird in den nächsten Testfällen der angelegte Artikel wieder gelöscht.

4.3.4 Artikel löschen

Wenn ein neues Teigprodukt angelegt wird, werden zwei Schritte vollzogen: Es wird ein Artikel angelegt mit der eingegebenen – oder auch ermittelten – Artikel- und Produktnummer, zeitgleich wird im System aber auch ein Produkt mit dieser Nummer angelegt. Will man nun diesen Artikel und das Produkt löschen muss das in einer definierten Reihenfolge geschehen. Zuerst die Löschung des Artikels, danach die Löschung des Produktes. Sollte man versuchen dies in gestürzter Reihenfolge durchzuführen, wird vom ERP-System eine Fehlermeldung ausgegeben, wie in Abbildung 19 ersichtlich:

Abbildung 16: Fehlermeldung bei falscher Löschreihenfolge (eigene Abbildung, 2023)

Man könnte hier die Funktionalität der richtigen Reihenfolge überprüfen, indem man zuerst versucht das Produkt zu löschen und bei der auf Seite 35 beschriebenen „MessageValidation“ auf diese Warnmeldung abfragt.

Der Testfall „Artikel löschen“ besteht aus vier Schritten, nach der Navigation auf die Detailseite des Artikels mit der ermittelten Artikelnummer wird der Menüpunkt „Löschen“ ausgewählt und die Nachfrage „Wollen Sie diesen Artikel wirklich löschen?“ mit „Ja“ bestätigt.

Step	Action	Field	Value
1	*Navigate to: EcoResProductDetailsExtended ("ecoresproductdetailsextended grid")*	**ItemId: BeginsWith**	=VALUE({{ItemId_1232}})+1
2	*Verwenden Sie den Filteroperator 'beginnt mit', um den Wert 'Artikelnummer' einzugeben.*		
3	*Klicken Sie auf 'Löschen'.*		
4	*Klicken Sie auf 'Ja'.*		

Tabelle 6: Testschritte "Artikel löschen"

Eine zukünftige Optimierung dieses Testfalls könnte das Abfragen der Systemnachricht „Artikel wurde gelöscht“ beinhalten. Auch eine Suche nach dem Artikel nach dessen Löschung kann hier zusätzlich für Sicherheit sorgen.

4.3.5 Produkt löschen

Der Testfall „Produkt löschen“ unterscheidet sich nur unwesentlich vom vorherigen Testfall „Artikel löschen“. Die Masken und die Funktionalität dieser sind analog aufgebaut, lediglich die Navigation zu den Produkten unterscheidet sich von der Navigation zu den Artikeln.

Step	Action	Field	Value
1	*Navigate to: EcoResProductListPage ("ecoresdistinctproductlistpage")*	**ItemId: BeginsWith**	=VALUE({{{ItemId_1232}}})+1
2	*Verwenden Sie den Filteroperator 'beginnt mit', um den Wert 'Artikelnummer' einzugeben.*		
3	*Klicken Sie auf 'Löschen'.*		
4	*Klicken Sie auf 'Ja'.*		

Tabelle 7: Testschritte "Produkt löschen"

Wieder wird auf die benötigte Maske navigiert, die zuvor ermittelte Artikel- und Produktnummer übergeben, und der Menüpunkt löschen ausgewählt. Die Nachfrage wird auch in diesem Testfall mit „Ja“ bestätigt.

Auch bei diesem Testfall könnte eine zukünftige Optimierung die Abfrage der Systemnachricht „Produkt wurde gelöscht“ beinhalten und es könnte durch eine spätere Suche nach dem gelöschten Produkt die Funktionalität der Löschung abgesichert werden.

Nach dem Testfall „Produkt löschen“ ist ein Durchlauf der Test Suite „Artikelanlage“ vollständig durchgeführt. Es wurde die höchste vergebene Produktnummer ermittelt, um eins erhöht und ein neuer Artikel mit verschiedenen Werten gefüllt, überprüft und gelöscht.

4.3.6 Vergleiche Dauer automatisierter Tests (AT) und manueller Tests

In den folgenden Tabellen werden die Zeiten von zehn durchgeführten Testdurchläufe der Test Suite „Artikelanlage“ miteinander verglichen. Die Reihenfolge bezieht sich dabei auf die bereits erwähnte Positionierung in der Test Suite:

1) Ermittlung der höchsten Artikelnummer
2) Artikel anlegen
3) Artikel prüfen
4) Artikel löschen
5) Produkt löschen

#	Testfall
1	Höchste Artikelnummer ermitteln
2	Artikel anlegen
3	Artikel prüfen
4	Artikel löschen
5	Produkt löschen

Tabelle 8: Testreihenfolge

Notiert wurden folgende Durchlaufarten:

- AT FullTest: Ist die Zeit – analog zu folgendem AT Full – die für die Abarbeitung der modularisierten Testfälle benötigt wird, ohne Hochladen der Ergebnisse in Azure DevOps.
- AT Full: Der komplette Durchlauf des automatischen Tests. Dies beinhaltet das Initialisieren der Test-Tools, die Testfälle selbst und das Hochladen der Ergebnisse in Azure DevOps. Durch die Modularisierung werden hier vor jedem Testfall die Tools initialisiert und nach jedem Testfall die Resultate hochgeladen.
- SW-Tester: Die manuelle Durchführung eines erfahrenen SW-Testers oder SW-Testerin. Bei diesen Aufzeichnungen wurden keine Fehleingaben gemacht, weiters wurden die Vergleichswerte aus dem Gedächtnis abgerufen.
- KeyUser: Die manuelle Durchführung eines KeyUsers der Fachabteilung. Diese sind zwar mit den Testfällen betraut, ab und an werden aber Fehleingaben getätigt. Die Vergleichswerte wurden auf einem Blatt Papier abgelesen, verglichen und abgehakt.

Im Unterschied zu den automatischen Testfällen wurden die manuellen Testdurchführungen nicht dokumentiert. Zwar wurden die Testfälle auf Korrektheit überprüft, jedoch werden die Ergebnisse aufgrund derzeit fehlender Testdokumentation nirgends gesammelt und gespeichert.

Um einen weiteren Vergleich zu schaffen wurden zusätzlich zu den oben beschriebenen Durchlaufarten zwei weitere Zeiten notiert, die zwar in Zukunft nicht zum Einsatz kommen, aber den Unterschied zwischen manuellen Tests und automatisierten Tests verdeutlichen sollen:

- AT TestOnly: Hier werden die fünf Testfälle zu einem zusammengefasst, ohne die Ergebnisse ins Azure DevOps hochzuladen. Durch das Zusammenfügen der einzelnen Testfälle zu einem großen Testfall spart man vier Mal das Initialisieren der Test-Tools.
- AT SingleUp: Wie bei AT TestOnly werden die fünf Testfalle zu einem zusammengefasst, jedoch werden die Ergebnisse dann in Azure DevOps hochgeladen und zentral gespeichert.

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:34		00:26	00:31		
2	01:30		01:26	02:07		
3	00:45		00:42	00:56		
4	00:19		00:25	00:30		
5	00:10		00:17	00:24		
Total	**03:19**	**06:15**	**03:16**	**04:28**	**01:05**	**02:37**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:38		00:25	00:30		
2	01:37		01:11	02:08		
3	00:42		00:46	00:55		
4	00:20		00:22	00:26		
5	00:11		00:13	00:23		
Total	**03:28**	**05:53**	**02:56**	**04:23**	**01:04**	**02:32**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:37		00:23	00:32		
2	01:32		01:10	02:09		
3	00:38		00:43	00:53		
4	00:20		00:23	00:32		
5	00:10		00:14	00:20		
Total	**03:18**	**05:52**	**02:53**	**04:25**	**01:03**	**02:35**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:35		00:22	00:32		
2	01:29		01:18	02:17		
3	00:36		00:46	01:02		
4	00:19		00:23	00:29		
5	00:10		00:14	00:23		
Total	**03:08**	**05:51**	**03:02**	**04:42**	**01:03**	**02:32**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:35		00:22	00:30		
2	01:27		01:06	02:09		
3	00:37		00:44	00:54		
4	00:19		00:21	00:30		
5	00:10		00:18	00:21		
Total	**03:07**	**06:15**	**02:52**	**04:24**	**01:03**	**02:31**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:35		00:20	00:32		
2	01:31		01:09	02:19		
3	00:39		00:46	01:02		
4	00:20		00:22	00:26		
5	00:10		00:14	00:24		
Total	**03:16**	**06:26**	**02:50**	**04:43**	**01:05**	**02:32**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:35		00:23	00:31		
2	01:31		01:00	02:00		
3	00:39		00:46	01:16		
4	00:20		00:21	00:31		
5	00:10		00:13	00:21		
Total	**03:16**	**07:45**	**02:44**	**04:39**	**01:05**	**02:34**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:36		00:21	00:30		
2	01:28		00:56	01:54		
3	00:37		00:42	01:05		
4	00:19		00:19	00:28		
5	00:10		00:16	00:21		
Total	**03:10**	**07:11**	**02:32**	**04:18**	**01:06**	**02:32**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:38		00:24	00:30		
2	01:30		00:58	02:24		
3	00:37		00:42	01:01		
4	00:19		00:19	00:27		
5	00:10		00:14	00:23		
Total	**03:14**	**06:03**	**02:37**	**04:44**	**01:05**	**02:36**

#	AT FullTest (in mm:ss)	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)	AT TestOnly (in mm:ss)	AT SingleUp (in mm:ss)
1	00:36		00:25	00:32		
2	01:34		01:02	01:55		
3	00:39		00:39	00:57		
4	00:20		00:19	00:30		
5	00:10		00:14	00:22		
Total	**03:19**	**05:46**	**02:40**	**04:17**	**01:08**	**02:33**

Tabelle 9: Zeiten der zehn durchgeführten Durchläufe „Artikel anlegen“

Bei allen der zehn Testdurchführungen benötigten die manuellen Tests weniger Zeit als der „AT Full“-Ansatz, bei dem die Werkzeuge vor jedem Schritt erst initialisiert werden müssen und nach jedem Schritt die Ergebnisse in Azure DevOps hochgeladen werden. Würde dieser Schritt auch bei den manuellen Tests erledigt werden, würde der automatische Test hier auch vorne liegen.

Werden nur die Tests durchgeführt ohne Speicherung der Ergebnisse – AT FullTest - kann man als erfahrener SW-Tester oder erfahrene SW-Testerin den automatisierten Test unterbieten, zu beachten sei aber, dass es bei der manuellen Durchführung zu keiner einzigen Fehleingabe oder Nachdenkpause kam. Bei diesem Ansatz wird keine Speicherung der Ergebnisse vorgenommen, kann also als direkter Vergleich zwischen manuellem und automatisierten Test herangezogen werden.

Durch Zusammenfassung der einzelnen Testfälle in einen großen Testfall, wie bei AT TestOnly und AT SingleUp, kann der automatisierte Test seine Stärke ausspielen. Auch als erfahrener SW-Tester oder erfahrene SW-Testerin benötigt man mindestens die doppelte Zeit, um sich durch das System zu navigieren und die Funktionen zu testen. Der große Vorteil hierbei ist, dass die Test-Tools nur einmal zu Beginn des Durchlaufs initialisiert werden und das zu testende System nur einmal aufgerufen. Jedoch hat dieser Ansatz auch einen gravierenden Nachteil, da mehrere Testfälle in einen verschmelzen und die Modularisierung verlorengeht. So müssen bei fehlschlagenden Testfällen

mehrere Funktionen oder Formulare überprüft werden, da diese verschiedenen Funktionen in einem Testfall zusammengefasst sind.

Zu beachten ist, dass das Initialisieren der Test-Tools, der Aufruf des zu testenden ERP-Systems, wie auch das Hochladen der Testergebnisse in etwa die Hälfte der benötigten Gesamtzeit in Anspruch nimmt. So wird bei der Test Suite „Artikel anlegen“ in etwa die Hälfte der benötigten Zeit beim Ansatz „AT Full“ für die Tool-Initialisierung aufgewendet.

Testfall	Tool-Init.	Gesamt	% Tool-Init
03:19	02:56	06:15	46,97%
03:28	02:25	05:53	41,13%
03:18	02:34	05:52	43,85%
03:08	02:43	05:51	46,30%
03:07	03:08	06:15	50,15%
03:16	03:10	06:26	49,29%
03:16	04:29	07:45	57,75%
03:10	04:01	07:11	55,93%
03:14	02:49	06:03	46,59%
03:19	02:27	05:46	42,36%

Tabelle 10: Anteil der Werkzeug-Initialisierung an der Gesamtzeit

Die nachfolgende Abbildung 16: Dauer Testfall / Tool-Init. stellt dieses Verhältnis noch einmal bildlich dar:

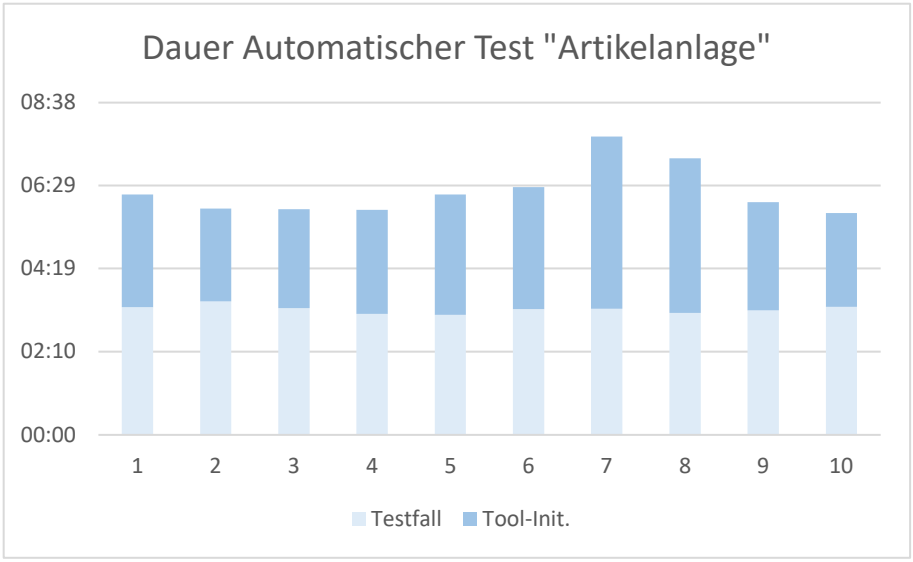

Abbildung 17: Dauer Testfall / Tool-Init. (eigene Abbildung, 2023)

Aber auch zwischen den Ansätzen „AT TestOnly“ und „AT SingleUp“ ist erkennbar, dass etwa 50% der Gesamtzeit für Tools, Navigation zu ERP-System und Hochladen der Ergebnisse aufgewendet wird.

Gesamt bietet sich folgendes Bild, das auch in Abbildung 17 noch mal verdeutlicht wird. Der in der Firma in Zukunft eingesetzte Ansatz „AT Full“ dauert pro Durchlauf zwar am längsten, beinhaltet aber die geeignetste Modularisierung und kümmert sich auch automatisiert um die zentrale Speicherung und Dokumentation der Testergebnisse.

Der manuelle Test kann bei den durchgeführten Testfällen zeitlich mithalten, jedoch wird die Automatisierung mit zunehmender Komplexität der Testfälle an der menschlichen Überprüfung vorbeiziehen. Unerreichbar hingegen ist die automatische Testdurchführung, bei der eine Funktionalität komplett in einen Testfall gepackt wird. Jedoch darf man den Zeitaufwand nicht vergessen, der bei fehlschlagenden Tests aufgewendet wird, um die fehlerhafte Programmierung zu suchen und finden.

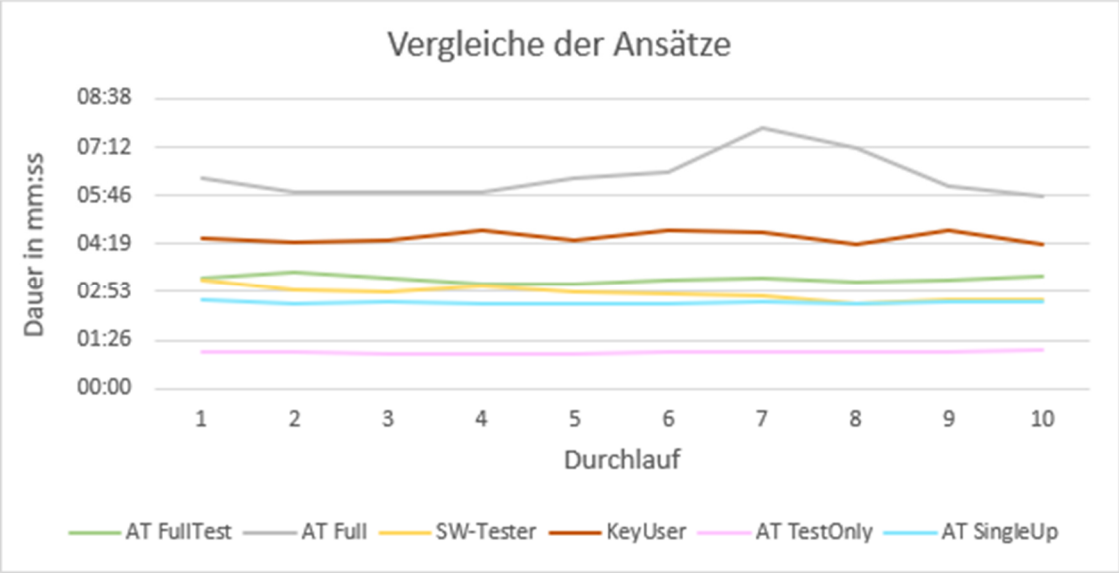

Abbildung 18: Zeitliche Vergleiche der Ansätze bei „Artikel anlegen“ (eigene Abbildung, 2023)

4.4 Testfall „Rohstoff bestellen“

Bei der Rohstoffbestellung wird überprüft, ob nach Neuanlage einer Bestellung mit einer Position der Lieferant korrekt übernommen wurde, ob der richtige Artikel in der Positionsliste erscheint, und ob das Bestelldatum, das Lieferdatum und die Menge des Artikels korrekt in die Bestellung übernommen wurden. Bei einer wirklichen Bestellung wird nach dessen Erstellung dem Lieferanten automatisch eine elektronische Benachrichtigung geschickt, dies wird in der Stage-Umgebung nur simuliert. Trotzdem wird ein Schalter „E-Mail gesendet“ auf dem Formular aktiviert. Auch dieser Schalter wird in dem Testfall überprüft.

Die Rohstoffbestellung passiert nur auf einer Maske und muss nach Anlage nicht gelöscht werden, deshalb ist dieser Testfall nicht in kleinere Testfälle aufgeteilt, sondern wird in einem durchgeführt.

In Tabelle 11 sind die Schritte des Testfalls zu sehen. Zuerst wird eine allgemeine Rechnung mit Lieferant und Daten erstellt, nach OK können die Artikelpositionen eingefügt werden. Dann werden diese Werte in Kopfzeile und Positionszeile überprüft.

Step	Action	Field	Value
1	*Navigate to: PurchTable ("purchtablelistpage")*		
2	*Klicken Sie auf 'Neu'.*		
3	*Geben Sie im Feld Kreditorenkonto einen Wert ein.*	**Kreditorenkonto**	300262
4	*Geben Sie im Feld 'Abschlussstichtag' ein Datum ein.*	**Abschlussstichtag**	=TODAY()+14
5	*Geben Sie im Feld 'Lieferdatum' ein Datum ein.*	**Lieferdatum**	=TODAY()+14
6	*Notieren Sie den Wert im Feld Kreditorenkonto {{OrderAccount_2061}}*		
7	*Notieren Sie den Wert im Feld Bestellung {{PurchId_2061}}*		
8	*Notieren Sie den Wert im Feld Lieferdatum {{DeliveryDate_2061}}*		
9	*Klicken Sie auf 'OK'.*		
10	*Geben Sie im Feld 'Artikelnummer' einen Wert ein.*	**Artikelnummer**	170-4843
11	*Geben Sie im Feld 'Menge' eine Zahl ein.*	**Menge**	10
12	*Überprüfen Sie, ob der Wert mit dem Wert übereinstimmt. {{DeliveryDate_2061}}*	**Validate Lieferdatum**	={{DeliveryDate_2061}}
13	*Überprüfen Sie, ob der Wert für 'Bestelldatum' heute ist.*	**Validate Bestelldatum**	=TODAY()
14	*Überprüfen Sie, ob der Wert für Artikelnummer '170-4843' ist.*	**Validate Artikelnummer**	170-4843
15	*Überprüfen Sie, ob der Wert für Menge '10' ist.*	**Validate Menge**	10
16	*Klicken Sie auf die Registerkarte 'Kopfzeile'.*		

17	*Überprüfen Sie, ob der Wert für 'Bestellung' mit dem Wert übereinstimmt. {{PurchId_2061}}*		
18	*Überprüfen Sie, ob der Wert für 'Kreditorenkonto' mit dem Wert übereinstimmt. {{OrderAccount_2061}}*		
19	*Überprüfen Sie, ob der Wert für Email gesendet 'TRUE' ist*	**Validate Email gesendet**	WAHR

Tabelle 11: Testfall "Rohstoff bestellen"

Dadurch, dass die Rohstoffbestellung auf einem Formular durchgeführt und am Ende nicht gelöscht wird und eben deswegen ein kurzer Testfall ist, wurde die Gegenüberstellung der Zeiten auf den automatischen Testdurchlauf und die manuellen Überprüfungen durch einen erfahrenen SW-Tester oder einer erfahrenen SW-Testerin und den KeyUser der Fachabteilung begrenzt. Da zusätzlich das Hochladen der Ergebnisse in Azure DevOps durch eine geringe Datenlast nicht ins Gewicht fällt, wurde auch dies nicht weiter aufgeschlüsselt.

Wie auch bei der Test Suite „Artikel anlegen“ wurden zehn Durchläufe pro Testart durchgeführt, wie in der folgenden Tabelle 12 angeführt ist:

#	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)
1	00:53	00:35	00:48
2	00:53	00:31	00:42
3	00:54	00:33	00:44
4	00:53	00:33	00:45
5	00:53	00:29	00:37
6	00:52	00:32	00:41
7	00:54	00:36	00:38
8	00:51	00:28	00:44
9	00:52	00:31	00:38
10	00:53	00:31	00:40

Tabelle 12: Zeiten der zehn durchgeführten Durchläufe "Rohstoff bestellen"

Auch bei dem kurzen Testfall „Rohstoffbestellung“ ist man mit einem manuellen Test schneller als die Testautomatisierung. Natürlich ist hier wieder zu berücksichtigen, dass bei einer Testautomatisierung die Ergebnisse am Ende in der Testdokumentation hinterlegt sind. Die Vergleiche der drei verwendeten Testansätze sind in der folgenden Abbildung 19 zu sehen.

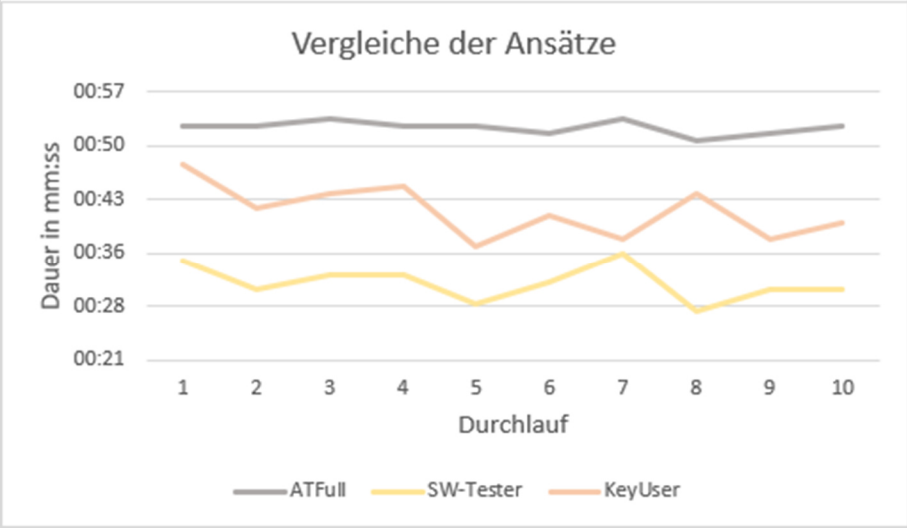

Abbildung 19: Zeitliche Vergleiche der Ansätze bei „Rohstoffbestellung“ (eigene Abbildung, 2023)

4.5 Testfall „Stückliste erstellen“

Bei diesem Testfall geht es darum eine Stückliste zu erstellen, die Informationen und Mengen der verwendeten Inhaltsstoffe enthält. Hier wird gleich das Verhältnis der Mengen (mit vier Nachkommastellen) der einzelnen Positionen zur Gesamtmenge im ERP-System berechnet, dieses wird in diesem Testfall mit einer Excel-Funktion überprüft. Am Ende wird die angelegte Stückliste auf der gleichen Formularmaske wieder gelöscht.

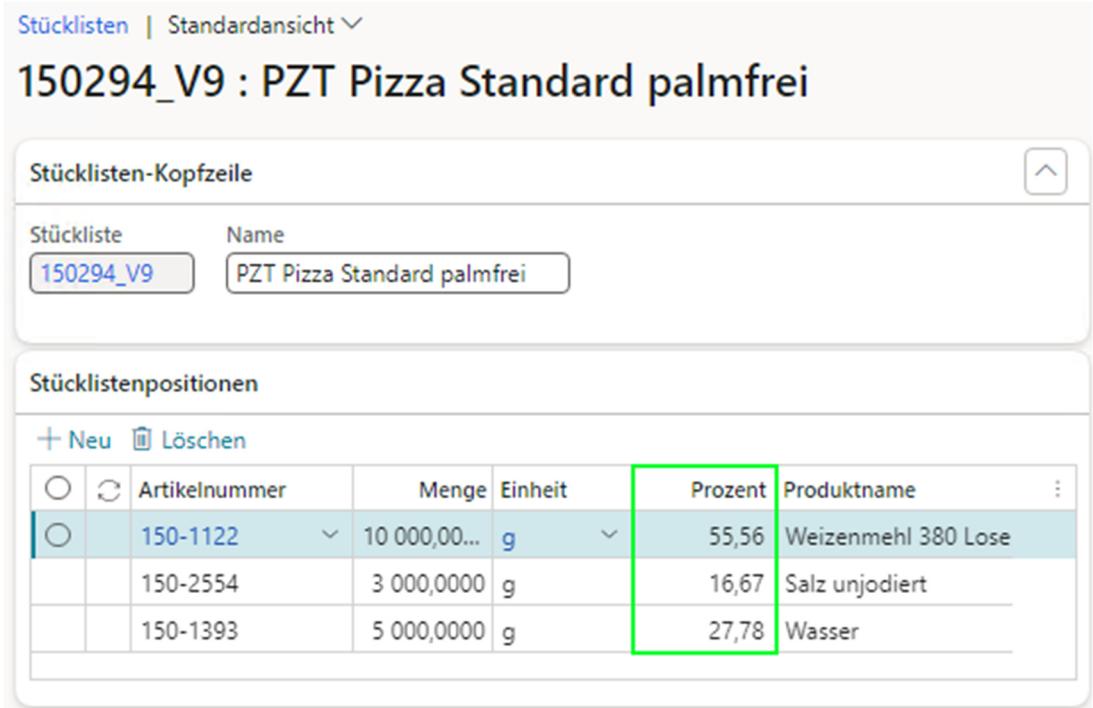

Abbildung 20: Vereinfachte Stückliste eines Pizzateiges (eigene Abbildung, 2023)

In Abbildung 20 sieht man die neu angelegte Stückliste mit der Nummer 150294. Da historisch bedingt bereits acht Versionen angelegt sind wird für den Testfall die neunte Version erstellt (V9). Da es sich um einen Pizzateig (PZT) handelt, wird der Name „PZT Pizza Standard palmfrei“ vergeben. Die Zutaten bei dieser Stückliste sind Weizenmehl, Salz und Wasser.

Step	Action	Field	Value
1	*Navigate to: BOMTable ("bomtable")*		
2	*Klicken Sie auf 'Neu'.*		
3	*Geben Sie im Feld Stückliste einen Wert ein.*	**Stückliste**	150294_V9
4	*Geben Sie im Feld Name einen Wert ein.*	**Name**	PZT Pizza Standard palmfrei
5	*Klicken Sie auf 'Neu'.*		
6	*Geben Sie im Feld 'Artikelnummer' einen Wert ein.*	**Artikelnummer**	150-1122
7	*Geben Sie im Feld 'Menge' eine Zahl ein.*	**Menge**	10000
8	*Klicken Sie auf 'Neu'.*		
9	*Geben Sie im Feld 'Artikelnummer' einen Wert ein.*	**Artikelnummer**	150-2554
10	*Geben Sie im Feld 'Menge' eine Zahl ein.*	**Menge**	3000
11	*Klicken Sie auf 'Neu'.*		
12	*Geben Sie im Feld 'Artikelnummer' einen Wert ein.*	**Artikelnummer**	150-1393
13	*Geben Sie im Feld 'Menge' eine Zahl ein.*	**Menge**	5000
14	*Wählen Sie in der Liste den gewünschten Datensatz aus.*	**GridBOM**	1
15	*Überprüfen Sie, ob der Wert für Menge kleiner 15000 ist.*	**Validate Menge less than 15000**	10000
16	*Überprüfen Sie, ob der Wert für Prozent '55.56' ist.*	**Validate Prozent**	55,56
17	*Wählen Sie in der Liste den gewünschten Datensatz aus.*	**GridBOM1**	2
18	*Überprüfen Sie, ob der Wert für Menge kleiner 10000 ist.*	**Validate Menge less than 10000**	3000
19	*Überprüfen Sie, ob der Wert für Prozent '16.67' ist.*	**Validate Prozent**	16,67

20	*Wählen Sie in der Liste den gewünschten Datensatz aus.*	**GridBOM2**	3
21	*Überprüfen Sie, ob der Wert für Menge kleiner 10000 ist.*	**Validate Menge less than 10000**	5000
22	*Überprüfen Sie, ob der Wert für Prozent '27.78' ist.*	**Validate Prozent**	27,78
23	*Klicken Sie auf 'Löschen'.*		
24	*Klicken Sie auf 'Ja'.*		

Tabelle 13: Testfall "Stückliste erstellen"

Bei den Testschritten 16, 19, und 22 werden die Prozente des ERP-Systems mit den Werten in der Spalte „Values“ überprüft. Diese sind aber keine fixen Werte, diese werden mittels Excel-Formel ROUND und den eingegebenen Mengenwerten ermittelt. Beispielweise wird der prozentuelle Anteil des eingesetzten Weizenmehls mit

=ROUND (D9 / (D9 + D12 + D15) * 100 , 2)

evaluiert. D9 bezieht sich hier auf die eingegebene Menge des Artikels 150-1122 (Weizenmehl), D12 auf die Menge von 150-2554 (Salz) und D15 auf die Menge des verwendeten Wassers – 150-1393. Da die prozentuellen Werte im ERP-System auf zwei Nachkommastellen begrenzt sind wird dies im Excel mit der ROUND-Funktion nachgestellt. Sollten bei den Mengen andere Werte benötigt werden, werden diese bei Schritt 7, 10, und 13 einfach ausgetauscht und die Excel-Funktion bezieht sich nun auf die neuen Werte. Natürlich beziehen sich auch die Überprüfungen der Mengen in den Schritten 15, 18, und 21 auf die ursprünglich eingegebenen Mengen.

Nach erfolgreicher Prüfung der Anteile der Inhaltsstoffe wird die Stückliste mittels Klick auf Löschen und Bestätigung der „Wollen Sie wirklich löschen?“-Nachricht wieder aus dem System entfernt.

Wie bei dem vorigen Testfall „Rohstoff bestellen“ wird die Stücklistenerstellung, -prüfung und -löschung auf einem Formular durchgeführt und durch wenige Testschritte in einen einzigen Testfall verpackt. Auch hier wurde die Gegenüberstellung der Durchlaufszeiten auf den automatischen Testdurchlauf und die manuellen Überprüfungen durch einen erfahrenen SW-Tester oder einer erfahrenen SW-Testerin und den KeyUsers der Fachabteilung begrenzt. Da auch bei der Stückliste das Hochladen der Ergebnisse in Azure DevOps durch eine geringe Datenlast nicht ins Gewicht fällt, wurde auch dies nicht weiter aufgeschlüsselt.

Wie bei den anderen Testfällen wurden zehn Durchläufe pro Alternative durchgeführt, wie in der folgenden Tabelle 14 zu sehen ist:

#	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)
1	01:01	00:49	01:06
2	01:00	00:47	01:03
3	01:04	00:43	01:10
4	00:59	00:45	01:04
5	01:02	00:45	01:04
6	01:01	00:44	01:02
7	01:00	00:43	01:05
8	01:00	00:43	01:07
9	01:02	00:46	01:03
10	01:01	00:43	01:02

Tabelle 14: Zeiten der zehn durchgeführten Durchläufe "Stückliste erstellen"

Wie auch beim vorherigen Testfall „Rohstoff bestellen“ ist man als erfahrener SW-Tester oder erfahrene SW-Testerin gegenüber dem automatischen Test im Vorteil. Jedoch ist bei diesem Testfall die Testautomatisierung den KeyUsers der Fachabteilungen ebenbürtig. Eine mögliche Ursache für dieses Ergebnis könnte die Distanz sein, die man mit der Maus pro Inhaltsstoff zurücklegen muss, auch der Wechsel zwischen Maus und Tastatur könnte hier nachteilig für manuelle Eingaben sein. Die Vergleiche der drei verwendeten Testansätze sind in der folgenden Abbildung 21 zu sehen.

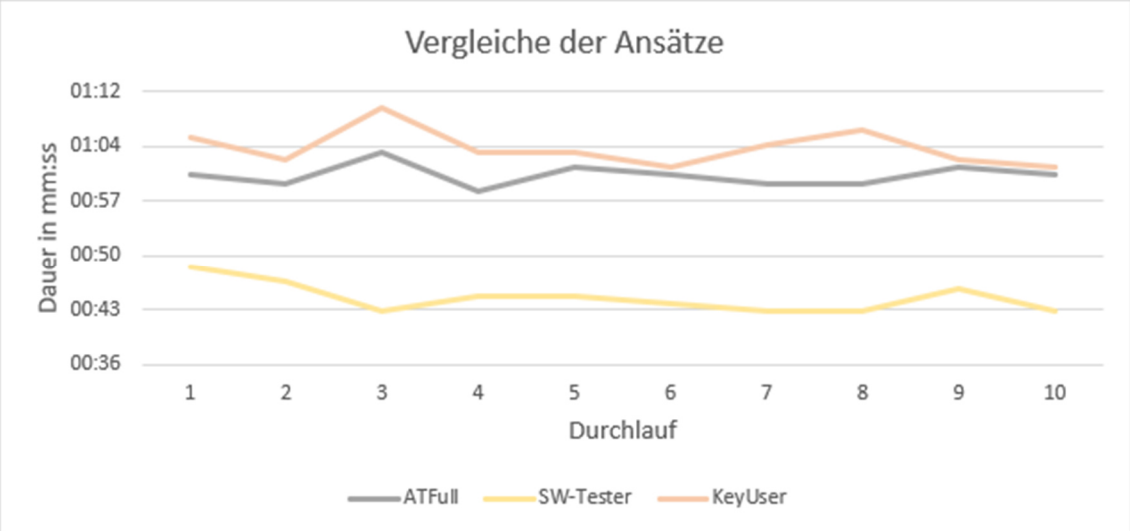

Abbildung 21: Zeitliche Vergleiche der Ansätze bei „Stückliste erstellen“ (eigene Abbildung, 2023)

4.6 Dokumentation und Speicherung in DevOps

Der große Vorteil beim automatisierten Testdurchlauf mit RSAT ist das Hochladen der Ergebnisse in Azure DevOps. Dort können die Test Suites, Testfälle und Testdurchläufe überprüft und gegebenenfalls Fehler lokalisiert werden.

Ein Beispiel eines korrekten Durchlaufs der Test Suite „Artikelanlage“ ist in der folgenden Abbildung 22 zu sehen. Die fünf Testfälle sind hier erfolgreich durchgelaufen, ersichtlich neben dem Testfalltitel sind die Ergebnisse, die verwendete Reihenfolge innerhalb der Test Suite und die Testfall-ID.

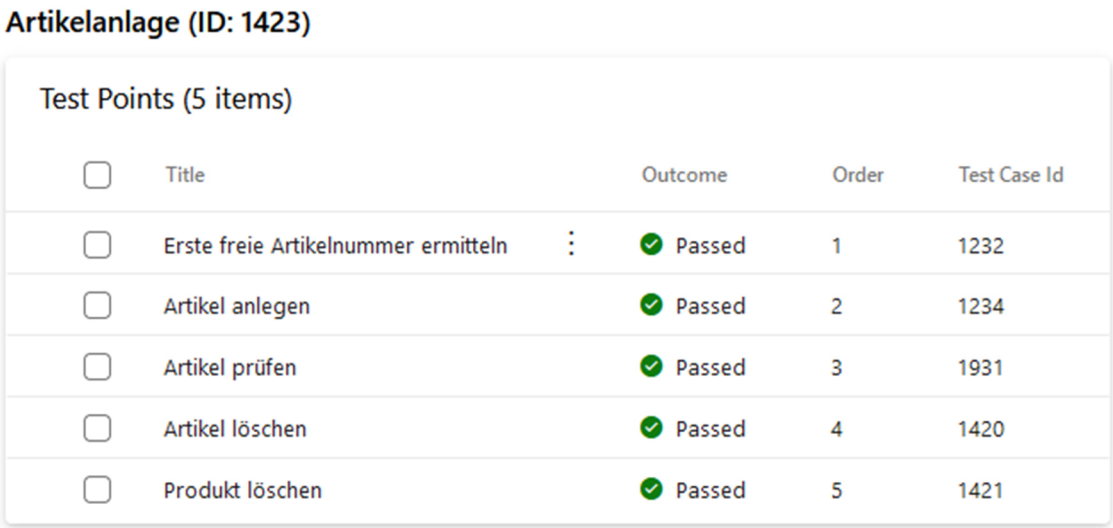

Abbildung 22: Dokumentierte Ergebnisse Test Suite "Artikelanlage" (eigene Abbildung, 2023)

Durch Auswahl eines Listeneintrages können weitere Informationen zu dem gewählten Testfall angezeigt werden, welche die durchlaufenen Schritte, die Systemumgebung und den Kontonamen zeigen. Dies zeige ich später bei fehlschlagenden Testfällen.

Sollten Testfälle fehlschlagen kann anhand der Aufschlüsselung rasch die Ursache für den nicht korrekten Durchlauf ermittelt werden. Dies werde ich in den folgenden zwei Testdurchläufen zeigen. Beim ersten Durchlauf mit Fehler wird versucht ein Produkt mit einer Nummer anzulegen, die aber im System schon vergeben ist. Beim zweiten fehlerhaften Durchlauf wird das Gewicht der Verpackung auf einen falschen Wert überprüft. Die Aufschlüsselung der beiden Durchläufe zeigen Abbildung 23 und Abbildung 25.

Artikelanlage (ID: 1423)

Test Points (5 items)

Title	Outcome	Order	Test Case Id
Erste freie Artikelnummer ermitteln	Passed	1	1232
Artikel anlegen	Failed	2	1234
Artikel prüfen	Not executed	3	1931
Artikel löschen	Not executed	4	1420
Produkt löschen	Not executed	5	1421

Abbildung 23: Fehler bei "Artikel anlegen" (eigene Abbildung, 2023)

Bei diesem Durchlauf wird zwar beim ersten Testfall noch die richtige höchste vergebene Nummer ermittelt, jedoch wird beim zweiten Testfall versucht, eben diese Nummer noch einmal im System anzulegen. Der Testfall schlägt fehl, die folgenden Testfälle werden nicht mehr durchgeführt, da kein Artikel angelegt werden konnte, den man nun prüfen und wieder löschen kann. Durch einen Klick auf die Zeile „Artikel anlegen“ wird die Aufschlüsselung des Testfalls aufgerufen. Dies zeigt die folgende Abbildung 24.

Run 1000997 - RSAT / Artikelanlage / Artikel anlegen

Summary

Failed

Run by	Claus Koizar
Test Plan	Artikelanlage
Test Case	Artikel anlegen
Configuration	Windows 10

Error message

```
Status: Test case failed.

(Details are also in the attached base log file Artikel_anlegen_SOL_1229_BaseLog.txt)

Steps:
Navigate to: EcoResProductDetailsExtended (&quot;ecoresproductdetailsextendedgrid&quot;)
Klicken Sie auf &#39;Neu&#39;.
Geben Sie im Feld Produktnummer einen Wert ein. (Value used in this run = &quot;160503&quot;)
        Warning: Die Nummer wurde bereits einem Produkt zugewiesen. Geben Sie eine neue Nummer ein.
Klicken Sie auf &#39;OK&#39;.
ERROR:
&lt;Message&gt;Click on the button OK (OKButton) has failed. It is disabled or not visible.
```

Abbildung 24: Error Log in Azure DevOps (eigene Abbildung, 2023)

Neben dem verwendeten Kontonamen des Testfalls, der Test Suite oder Test Plan und dem Testfall wird die Fehlermeldung und die fehlerhafte Stelle ausgegeben. Leider hat

Microsoft hier Probleme bei Sonderzeichen und Umlauten, nach einer kurzen Einarbeitungszeit kann die Fehlermeldung aber auch so rasch verstanden werden.

Es wurde bei diesem Testdurchlauf versucht einen Artikel mit der Nummer 16053 anzulegen, jedoch war bei den durchgeführten Tests 16053 immer die letzte vergebene Nummer. Für einen korrekten Durchlauf müsste die Nummer 16054 betragen.

Das ERP-System erkennt hier, dass ein Artikel mit einer bereits vergebenen Nummer angelegt werden soll und gibt eine Info-Meldung an:

Warning: Die Nummer wurde bereits einem Produkt zugewiesen. Geben Sie eine neue Nummer ein.

Durch die systeminterne Überprüfung wird der OK-Button zur Anlage eines Produkts gesperrt. Genau dies lässt den Durchlauf auch fehlschlagen:

ERROR: Click on the button OK (OKButton) has failed. It is disabled or not visible.

Bei diesem Durchlauf ist man mit einem manuellen Test schneller beim Erkennen der Fehlersituation. Bei Hinweis, dass diese Nummer bereits vergeben wurde kann eine Person den Test abbrechen, die Testautomatisierung ignoriert diesen Hinweis und versucht die weiteren Schritte abzuarbeiten.

Beim zweiten fehlerhaften Durchlauf wird ein Produkt mit dem Packungsgewicht 275g angelegt, jedoch wird auf 375g überprüft. Wie auch beim ersten Durchlauf werden die nachfolgenden Tests dann nicht mehr ausgeführt. Wie vorhin erwähnt zeigt die Abbildung 25 den zweiten fehlerhaften Testdurchlauf.

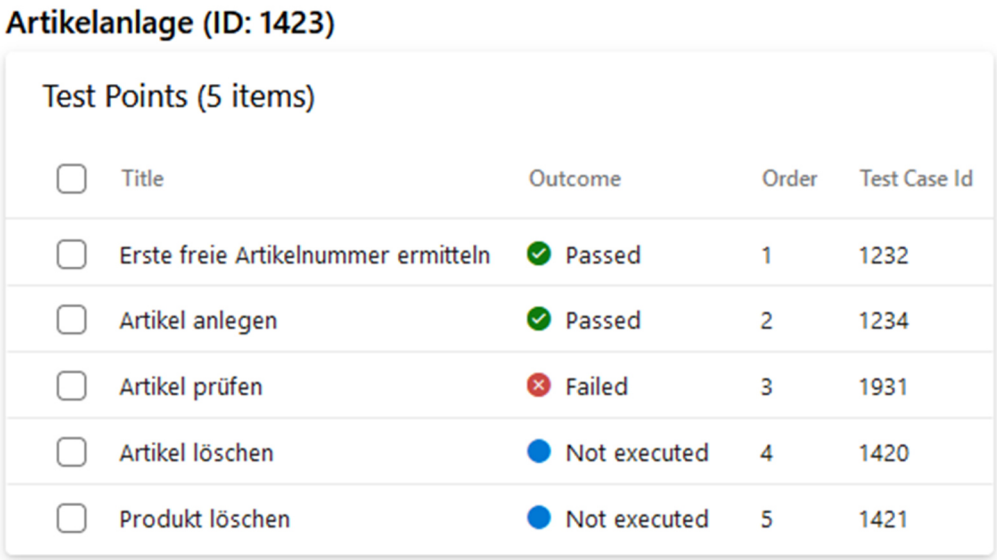

Abbildung 25: Fehler bei "Artikel prüfen" (eigene Abbildung, 2023)

Auch hier wird durch Klick auf die Zeile mit dem fehlerhaften Testfall dessen Abarbeitung angezeigt. Die folgende Abbildung 26 zeigt die Ausführung von „Artikel prüfen“ und die fehlerhafte Vergleichsoperation. (Aus Platzgründen wurden die funktionierenden Vergleiche ausgeblendet)

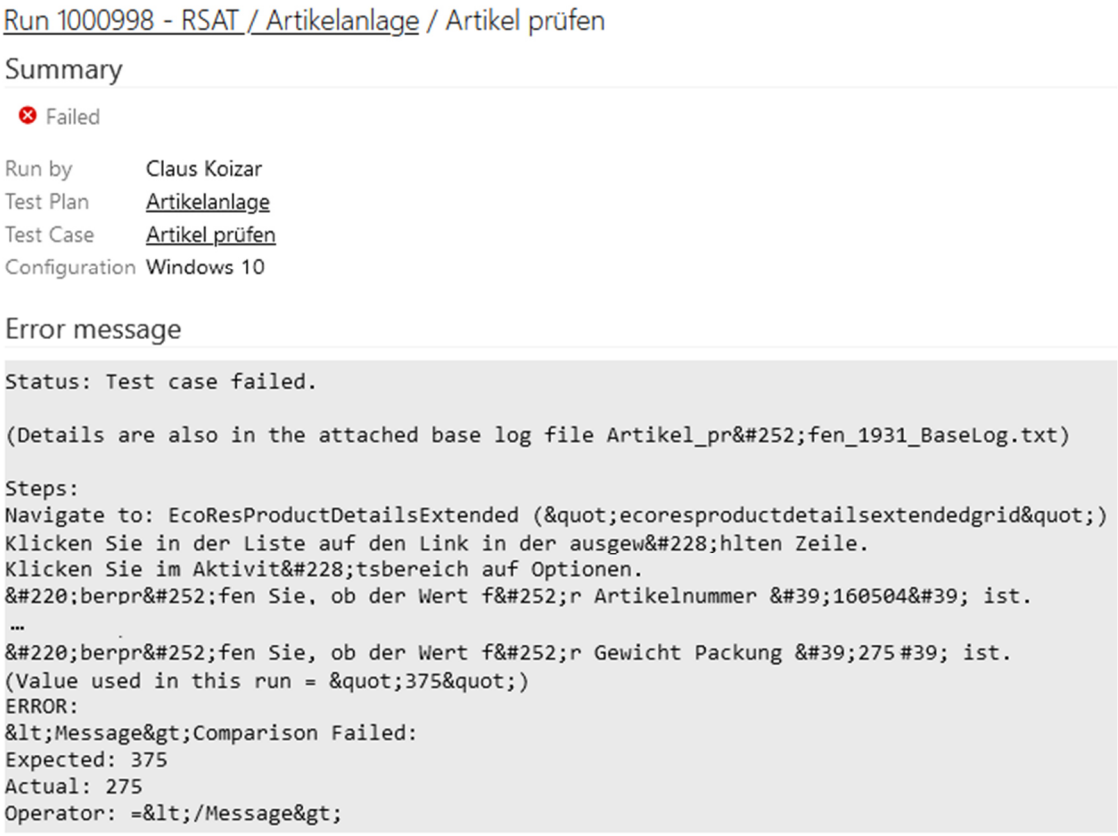

Abbildung 26: Error Log in Azure DevOps (eigene Abbildung, 2023)

Der fehlerhafte Bereich zeigt zuerst „Überprüfen Sie, ob der Wert für Gewicht Packung 275 ist. Der verwendete Wert aber ist 375. So wird ein Fehler ausgeworfen:

ERROR: Comparison Failed:

Expected: 375

Actual: 275

Operator: =

Damit ist auf den ersten Blick ersichtlich, bei welcher Aktivität oder Überprüfung der Testfall gescheitert ist. Der automatische Test ist bei Vergleichsoperationen schneller als manuelle Tests, da RSAT nicht scrollen muss und auch beim Vergleichen zweier Werte Vorteile gegenüber Testern und Testerinnen besitzt.

Auch bei den anderen beiden Testfällen „Rohstoff bestellen“ und „Stückliste erstellen“ können auf die oben gezeigte Art Fehler angezeigt und lokalisiert werden. Durch die

Kürze dieser Testfälle unterscheiden sich die Zeiten zwischen automatisierten und manuellen Tests aber nur unwesentlich – analog zu den erfolgreich durchgeführten Testfällen.

Um die Durchläufe grafisch aufzubereiten gibt es im Testmanagement von Azure DevOps die Möglichkeit Diagramme der Testdurchführungen anzeigen zu lassen. So kann beispielsweise nachvollzogen werden, wie viele Testfälle in den durchgeführten Überprüfungen fehlschlagen. In Abbildung 27 werden zwei Diagramme der Test Suite „Artikelanlage“ angezeigt, die bei diesem Durchlauf zwei erfolgreiche, einen fehlgeschlagenen Test und zwei nicht durchgeführte Tests beinhalten. Die Diagramme sind die grafische Darstellung des Durchlaufs bei Abbildung 25.

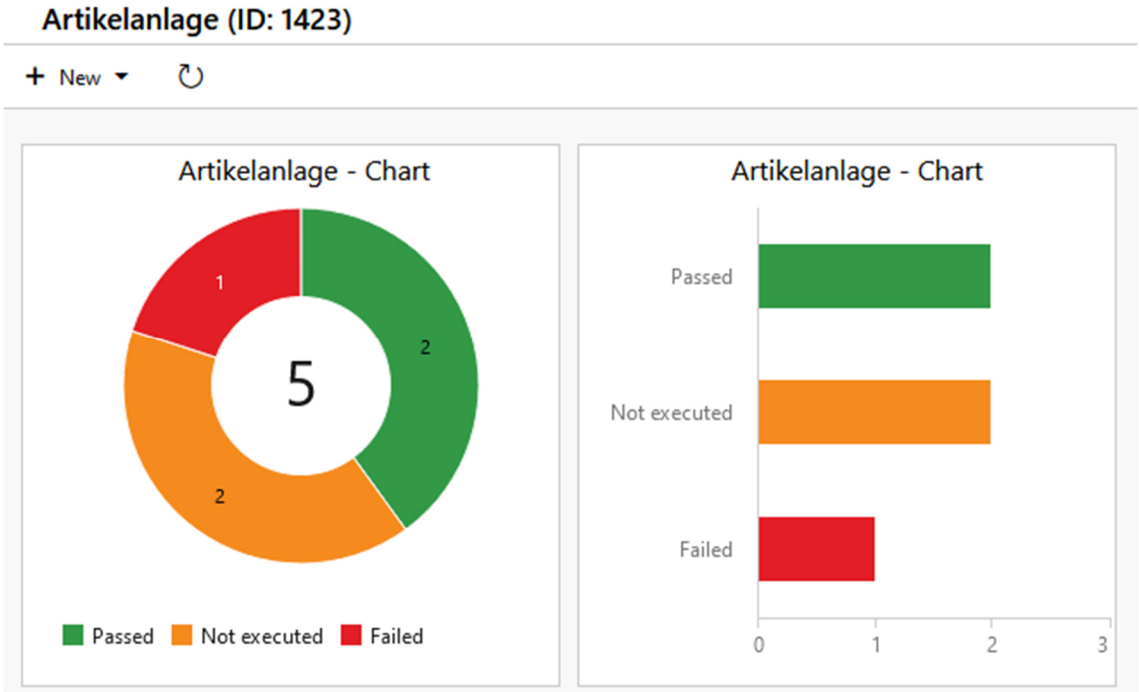

Abbildung 27: Diagramme von "Artikelanlage" (eigene Abbildung, 2023)

4.7 Modultest „Einheitenumrechnung“ (UnitConversion_Test)

Beim Unit Test oder Modultest „Einheitenumrechnung“ wird überprüft, ob die Umrechnung von Kartons auf Paletten und Stück auf Kartons sinnvolle Werte enthält. Da jeder Artikel unterschiedliche Dimensionen besitzt und auch Kartons unterschiedliche Größen haben, liegt der Füllfaktor hier zwischen vier und 29 Stück. Da aber auch einzelne Muster der Teige verschickt werden können, muss die Abfrage bei Stück auf Kartons zwischen eins und 30 liegen. Bei der Umrechnung zwischen Paletten und Kartons liegen die erlaubten Werte zwischen 30 und 200 Kartons pro Palette.

```
[TestMethod()]
public void UnitConversion_Test()
{
  InventTable inventTable;
  UnitOfMeasure UOM1, UOM2;
  UnitOfMeasureConversion uOMC;

  while select uOMC
  join inventTable where uOMC.Product == inventTable.Product
  && inventTable.ItemId like '1__0____'
  join UOM1 where uOMC.FromUnitOfMeasure == UOM1.RecId
  join UOM2 where uOMC.ToUnitOfMeasure == UOM2.RecId
  {
    if(!strCmp(UOM1.Symbol, 'Kar')&& !strCmp(UOM2.Symbol, 'Stk')
       && (uOMC.Factor < 1 || uOMC.Factor > 29))
    {
      this.assertNotNull(null, strFmt('Wrong value detected -
      ItemId: %1, From: %2, To: %3, Number: %4',
      inventTable.ItemId, UOM1.Symbol, UOM2.Symbol,
      uOMC.Factor));
    }
    if(!strCmp(UOM1.Symbol, 'Pal')&& !strCmp(UOM2.Symbol, 'Kar')
       && (uOMC.Factor < 30 || uOMC.Factor > 200))
    {
      this.assertNotNull(null, strFmt('Wrong value detected -
      ItemId: %1, From: %2, To: %3, Number: %4',
      inventTable.ItemId, UOM1.Symbol, UOM2.Symbol,
      uOMC.Factor));
    }
  }
}
```

Zuerst werden die Verkaufsartikel aus der Produkttabelle geladen inklusive deren Umwandlung in den Umwandlungstabellen. Da die Relation hier aus „von Verpackung“ und „zu Verpackung“ besteht, muss die Tabelle „UnitOfMeasure“ zweimal an die

Haupttabelle verknüpft werden. Besteht die Einheit jetzt aus „Karton“ und „Stück“, wird abgefragt, ob sich der Wert unter eins oder größer 29 befindet. Ist die Umwandlung zwischen „Palette“ und „Karton“, wird auf kleiner 30 und größer 200 abgefragt. In den Abfragen befindet sich der Trigger, der die Fehlermeldung ausgibt. Bei diesem Beispiel wird durch die „assertNotNull“-Abfrage auf null auf jeden Fall ein Fehler ausgeworfen, wenn die Abfrage betreten wird.

Zu Testzwecken wurde ein Artikel mit der Nummer 10100268 angelegt, bei dem die falsche Belegung gilt: In einen Karton passen 30 Stück. Da dies außerhalb der festgelegten Grenze von 29 Stück liegt meldet der Unit Test einen Fehler, wie Abbildung 28 zeigt.

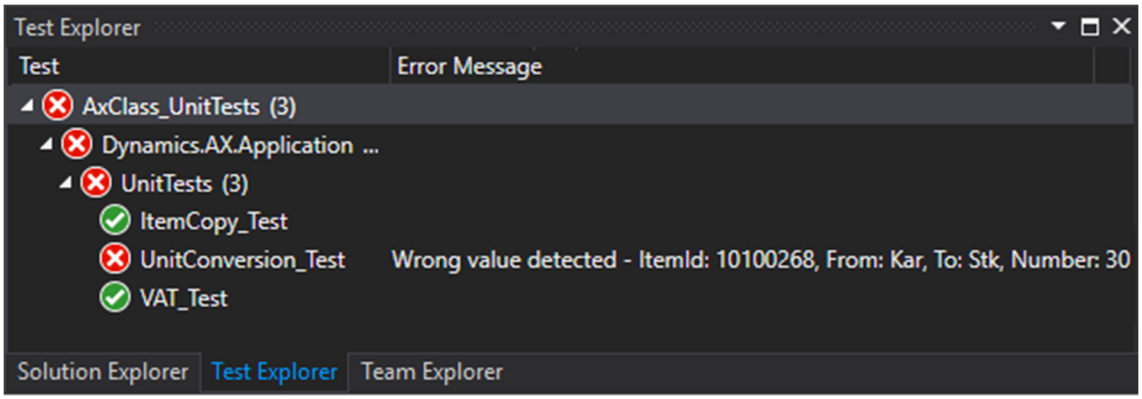

Abbildung 28: Fehlgeschlagene Umwandlung von "Karton" zu "Stück" (eigene Abbildung, 2023)

Der selbe Artikel hat auch zu Testzwecken eine falsche Belegung zwischen „Paletten“ und „Kartons“. Hier wird diese mit 0 angegeben, auch das liegt außerhalb der Grenze der zweiten Abfrage, sodass ein Fehler gemeldet wird, wie in Abbildung 29 gezeigt wird.

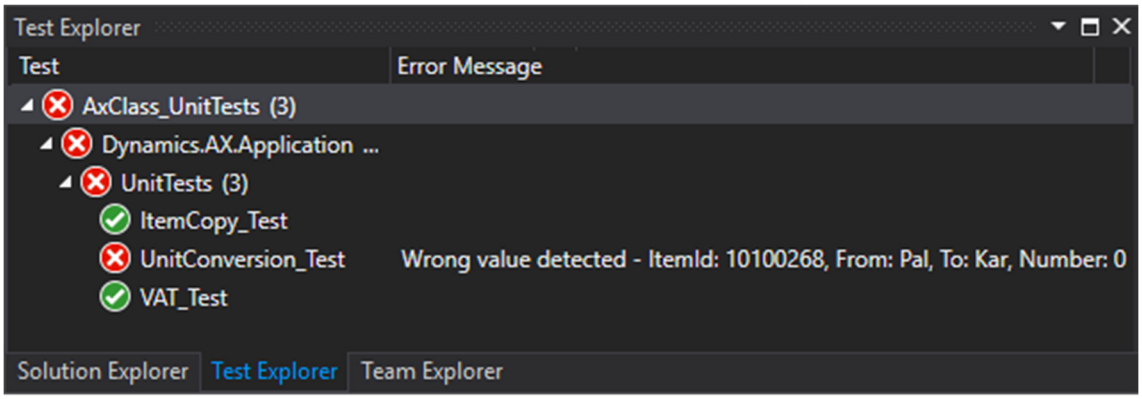

Abbildung 29: Fehlgeschlagene Umwandlung von "Palette" zu "Kartons" (eigene Abbildung, 2023)

4.8 Modultest “Mehrwertsteuer ermitteln“ (VAT_Test)

Bei „Mehrwertsteuer ermitteln“ werden die vorhandenen Steuersätze auf die korrekten Werte überprüft. Es werden verschiedene Sätze von 5% bis 27% auf den korrekten Wert

und die korrekte Bezeichnung überprüft. Ein Spezialfall ist der Eintrag „FREI“, der für steuerfreie Intercompany-Lieferungen zwischen der Produktionsstätte in Celldömölk, Ungarn und Sollenau, Österreich verwendet wird.

```
[TestMethod()]
public void VAT_Test()
{
  TaxGroupData taxGroupData;
  TaxValue taxValue;
  int ii;

  container types = ['E5', 'E10', 'E20', 'FREI', 'U5', 'U10',
             'U18', 'U20', 'U27', 'V10', 'V12', 'V13', 'V20'];
  container value = [5, 10, 20, 0, 5, 10, 18,
                     20, 27, 10, 12, 13, 20];
  while select taxGroupData
  {
    ii = conFind(types, taxGroupData.TaxCode);
    if(ii > 0)
    {
      taxValue = TaxData::percent(taxGroupData.TaxCode);
      this.assertEquals(taxValue, conPeek(value, ii),
      strFmt('Wrong value detected - Type: %1, Expected: %2,
      Returned: %3', conPeek(types, ii), conPeek(value, ii),
      taxValue));
    }
  }
}
```

Die verschiedenen Steuersatz-Bezeichnungen und ihre zugehörigen Werte werden in zwei Containern aufbewahrt. Die Tabelle taxGroupData mit den Steuerinformationen wird nach diesen Bezeichnungen (Typen) durchsucht und bei erfolgreicher Suche wird der Wert in der Tabelle mit den dementsprechenden Werten im Container verglichen. An vierter Stelle ist der Typ „FREI“ zu finden, der als Wert – ebenfalls an vierter Stelle

– 0 besitzt. In der Abfrage werden die Werte verglichen und bei Ungleichheit ein Fehler ausgeworfen.

In den folgenden zwei Abbildungen 30 und 31 sind fehlerhafte Einträge in der Tabelle zu sehen. Beim ersten Fehlerfall ist bei „FREI“ eine Zahl > 0 angegeben. Dies würde bewirken, dass steuerfreie Lieferungen fälschlicherweise einen Steueraufschlag bekämen. Das automatisierte Testsystem erkennt den Fehler und gibt einen Fehler mit den falschen Werten zurück.

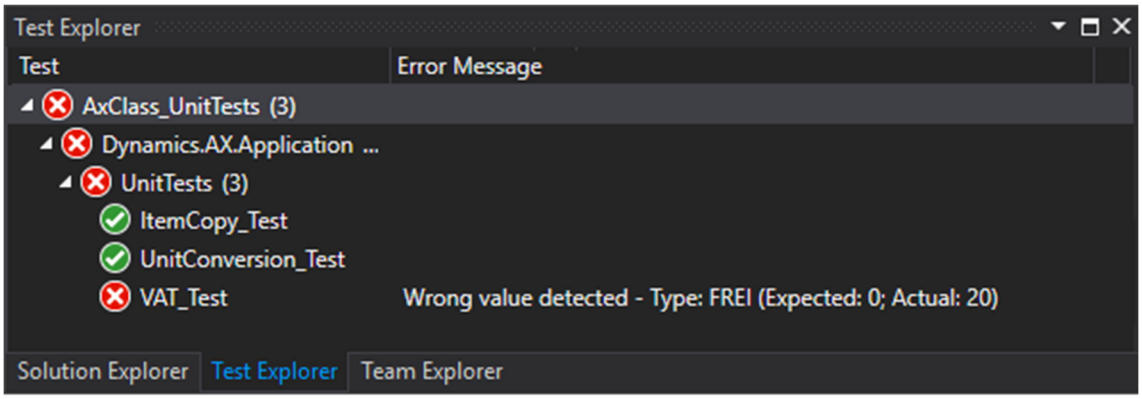

Abbildung 30: Fehlgeschlagene "FREI"-Überprüfung (eigene Abbildung, 2023)

Beim zweiten Fehlerfall wird ein anderer falscher Mehrwertsteuersatz vom System gemeldet. Hier wurde fälschlicherweise bei E20 ein Satz von 10% eingetragen, dieser müsste aber 20% betragen. Auch hier wird ein fehlgeschlagener Durchlauf zurückgeliefert.

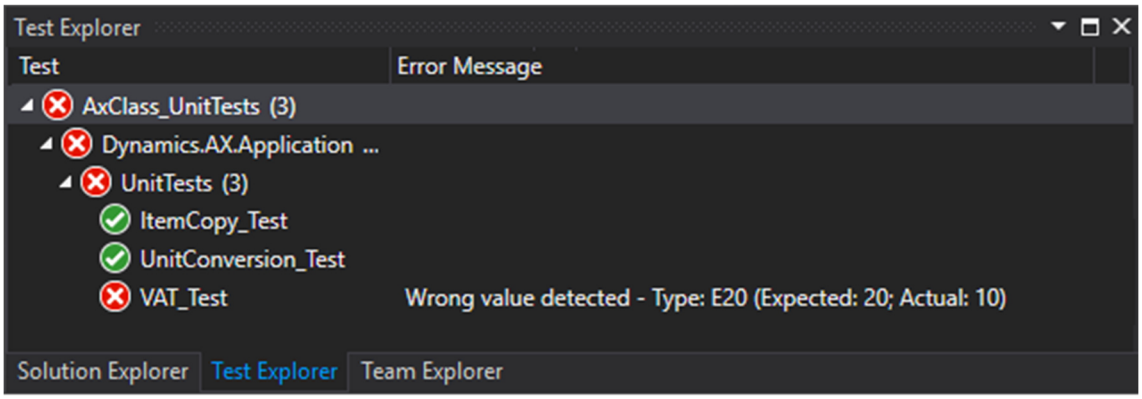

Abbildung 31: Fehlgeschlagene "E20"-Überprüfung (eigene Abbildung, 2023)

4.9 Modultest „Artikel kopieren“ (ItemCopy_Test)

Beim „Artikel kopieren“ wird eine systemeigene Funktion „buf2Buf“ überprüft, die ein Objekt klont. Im Testfall wird nach dem Klonen die Artikelnummer auf eine freie Artikelnummer gesetzt, da wie aus der Test Suite „Artikelanlage“ bekannt keine doppelte Artikelnummer angelegt werden kann. Danach werden die geklonten Werte überprüft.

```
[TestMethod()]
public void ItemCopy_Test()
{
  InventTable inventTable_old = InventTable::find('10100344');
  InventTable inventTable_new;

  buf2Buf(inventTable_old, inventTable_new);
  inventTable_new.ItemId = '10100345';

  this.assertNotEqual(inventTable_old.ItemId,
                      inventTable_new.ItemId, 'Same ItemId!');
  this.assertEquals(inventTable_old.NameAlias,
                    inventTable_new.NameAlias, 'Name wrong!');
  this.assertEquals(inventTable_old.Depth,
                    inventTable_new.Depth, 'Depth wrong!');
  this.assertEquals(inventTable_old.Height,
                    inventTable_new.Height, 'Height wrong!');
  this.assertEquals(inventTable_old.DataAreaId,
                    inventTable_new.DataAreaId, 'Area wrong!');
}
```

Zuerst wird der vorhandene Artikel mit der Artikelnummer 10100344 im System gesucht, von diesem ein Klon angelegt und dessen Nummer auf eine freie Nummer gesetzt. Anschließend werden die Artikeldimensionen des Klons mit dem Original verglichen. Zu beachten ist, dass im Gegensatz zu den restlichen Abfragen bei den Artikelnummern abgefragt wird, ob diese unterschiedlich sind.

Beim folgendem fehlerhaften Testfall wird die Artikelnummer durch einen Fehler nicht umgesetzt, dies zeigt Abbildung 32:

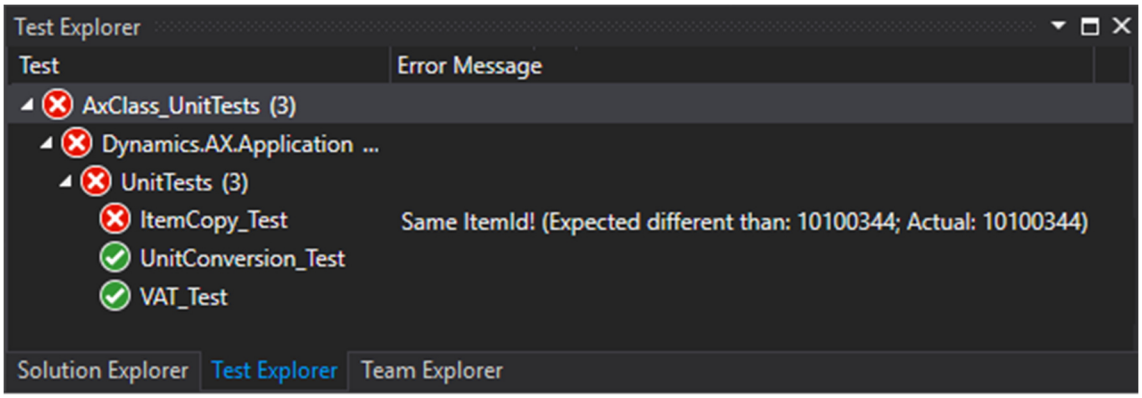

Abbildung 32: Fehlerhafte, doppelte Artikelnummer (eigene Abbildung, 2023)

Beim zweiten fehlerhaften Testfall wird die Tiefe des Originalartikels durch einen internen Fehler beim Klonen nicht kopiert. Dieser Wert bleibt beim neuen Artikel auf 0, somit ist die Kopie nicht erfolgreich durchgelaufen und das System meldet hier einen Fehler. Dies zeigt Abbildung 33.

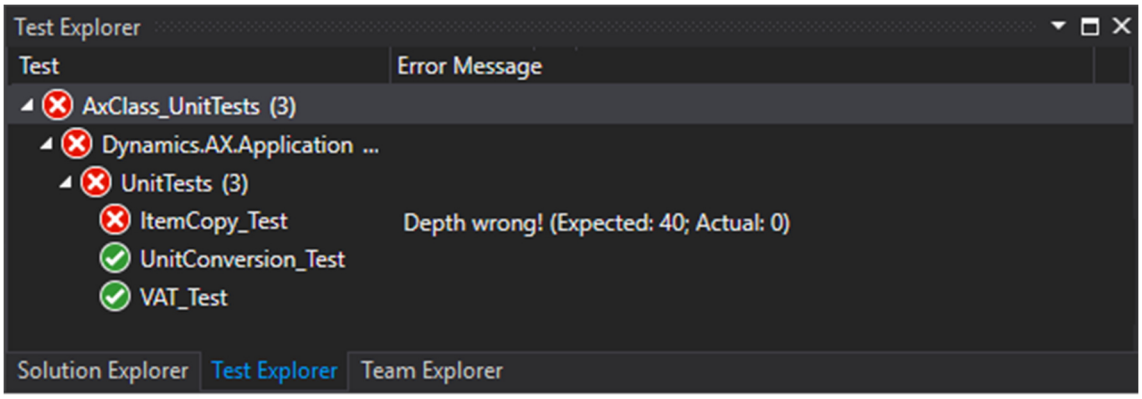

Abbildung 33: Fehlerhafte Tiefe bei der Artikelkopie (eigene Abbildung, 2023)

4.10 Vergleiche Dauer automatisierter Tests (AT) und manueller Tests - Modultests

Nach der Durchführung der Modultests wird die Dauer der einzelnen Modultests in der Entwicklungsumgebung angegeben, diese bewegen sich im Bereich zwischen zwei und drei Sekunden. Jedoch muss man auch hier wieder das Initialisieren der Testumgebung und der Werkzeuge hinzuzählen. Beim lokalen Testdurchlauf beträgt diese Initialisierung im Durchschnitt eine Minute, beim Aufruf in der Build-Pipeline beträgt das Initialisieren im Durchschnitt drei Minuten. Die Abbildung 34 zeigt eine erfolgreiche Durchführung der drei Modultests, die Dauer und die benötigte Gesamtzeit für alle der durchgeführten Modultests.

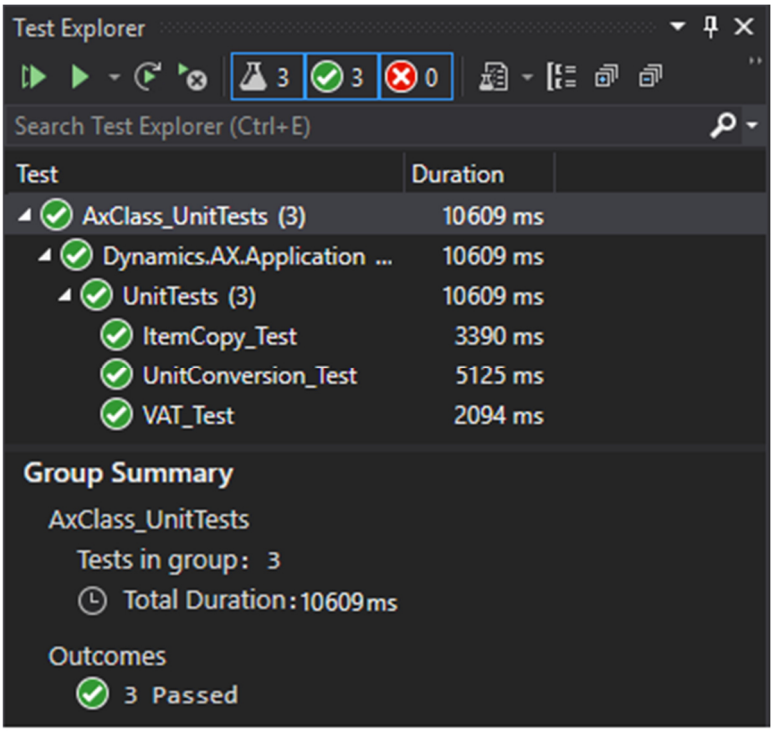

Abbildung 34: Dauer der Modultests (eigene Abbildung, 2023)

Da die Modultests ohne grafische Benutzeroberfläche und Frontend durchgeführt werden haben sie einen enormen zeitlichen Vorteil gegenüber manueller Kontrolle der durchgeführten Testfälle. Während sich die Überprüfung der Steuersätze auf 13 Einträge beschränkt, die manuell im Durchschnitt in 4,5 Minuten und automatisiert in zwei Sekunden überprüft werden können, sind es bei der Einheitenumrechnung schon 200 Datensätze in der Artikeltabelle, die über zwei Einträge in der zugehörigen Umrechnungstabelle verfügen. Hier beläuft sich die automatisierte Überprüfung auf durchschnittlich fünf Sekunden und die manuelle Überprüfung der 200 Datensätze liegt zwischen 50 und 60 Minuten. Die Artikelkopie wird vom automatischen Testsystem durchschnittlich in 3,5 Sekunden durchlaufen, ein manueller Test wird hier durchschnittlich in 3 Minuten durchgeführt.

Nachfolgend ist eine Auflistung der Modultests und dem Vergleich zwischen manueller und automatisierter Tests. Wie bei den durchgeführten Tests in RSAT wird auch hier jeder der drei Modultests zehn Mal durchlaufen.

Da bei den Modultests nur einzelne Funktionen überprüft werden, wird auch bei diesen Tests das einmalige Initialisieren der Testwerkzeuge pro Testlauf mitgestoppt. Hierbei ist zu beachten, dass diese in der Build-Pipeline nur einmal initialisiert werden.

Tabelle 15 und Abbildung 35 zeigen die zehn Testdurchläufe des Modultests „Einheitenumrechnung“.

#	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)
1	01:05	53:27	60:20
2	01:03	49:00	63:01
3	01:05	51:05	58:44
4	01:06	52:57	67:58
5	01:05	48:31	64:40
6	01:05	53:34	60:30
7	01:05	50:50	62:36
8	01:03	50:11	63:45
9	01:05	49:33	59:44
10	01:06	47:02	62:29

Tabelle 15: Zeiten der zehn durchgeführten Durchläufe "Einheitenumrechnung"

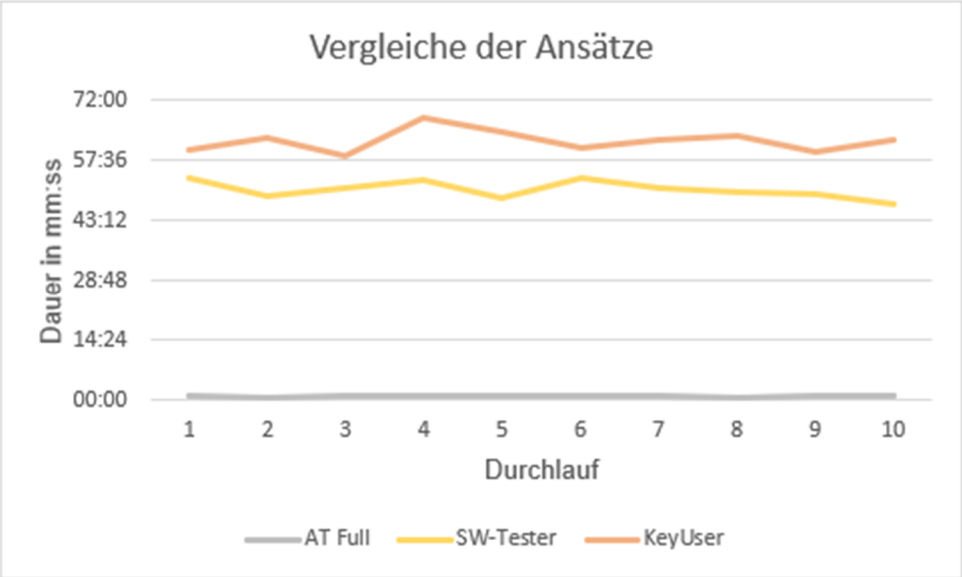

Abbildung 35: Zeitliche Vergleiche der Ansätze bei „Einheitenumrechnung“ (eigene Abbildung, 2023)

Bei Modultests mit größeren Datenmengen, verknüpften Tabellen oder verzweigten Formularen oder Prozessen auf der ERP-Oberfläche zeigt sich die Zeiteinsparung durch automatisierte Tests deutlich. Beim manuellen Test muss zuerst das Artikelformular aufgerufen und danach zur Einheitenumrechnung navigiert werden. Dies geschieht beim automatisierten Test mit einer einzigen Anweisung.

Tabelle 16 und Abbildung 36 zeigen die zehn Testdurchläufe des Modultests „Mehrwertsteuer ermitteln“.

#	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)
1	01:02	04:54	05:34
2	01:02	04:36	05:12
3	01:02	04:17	05:13
4	01:04	04:28	05:22
5	01:02	04:36	05:19
6	01:02	04:40	05:19
7	01:01	04:39	05:20
8	01:07	04:27	05:31
9	01:02	04:19	05:24
10	01:02	04:23	05:32

Tabelle 16: Zeiten der zehn durchgeführten Durchläufe "Mehrwertsteuer ermitteln"

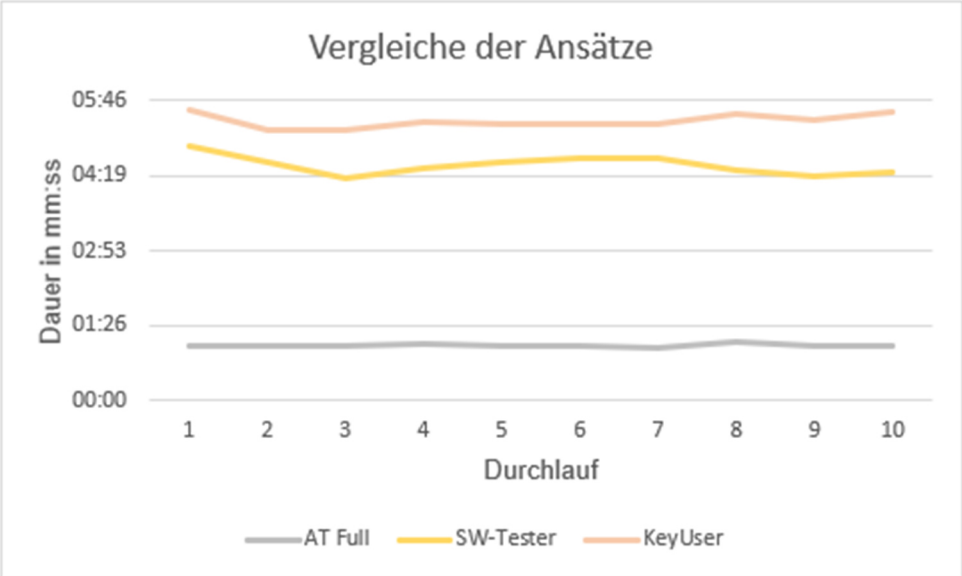

Abbildung 36: Zeitliche Vergleiche der Ansätze bei „Mehrwertsteuer ermitteln“ (eigene Abbildung, 2023)

Auch beim Modultest „Mehrwertsteuer ermitteln“ ist eine Zeitersparnis ersichtlich, jedoch sind die Steuersätze auch beim manuellen Test auf einer Seite übersichtlich dargestellt. Deswegen kann im Gegensatz zum vorherigen Modultest weniger Zeit eingespart bleiben.

Tabelle 17 und Abbildung 37 zeigen die zehn Testdurchläufe des Modultests „Artikel kopieren“.

#	AT Full (in mm:ss)	SW-Tester (in mm:ss)	KeyUser (in mm:ss)
1	01:03	03:05	03:47
2	01:03	03:01	03:35
3	01:05	02:56	03:39
4	01:02	03:00	03:11
5	01:05	02:40	03:40
6	01:01	02:53	03:28
7	01:05	02:33	03:11
8	01:03	03:11	03:17
9	01:05	02:53	03:46
10	01:03	02:44	04:02

Tabelle 17: Zeiten der zehn durchgeführten Durchläufe "Artikel kopieren"

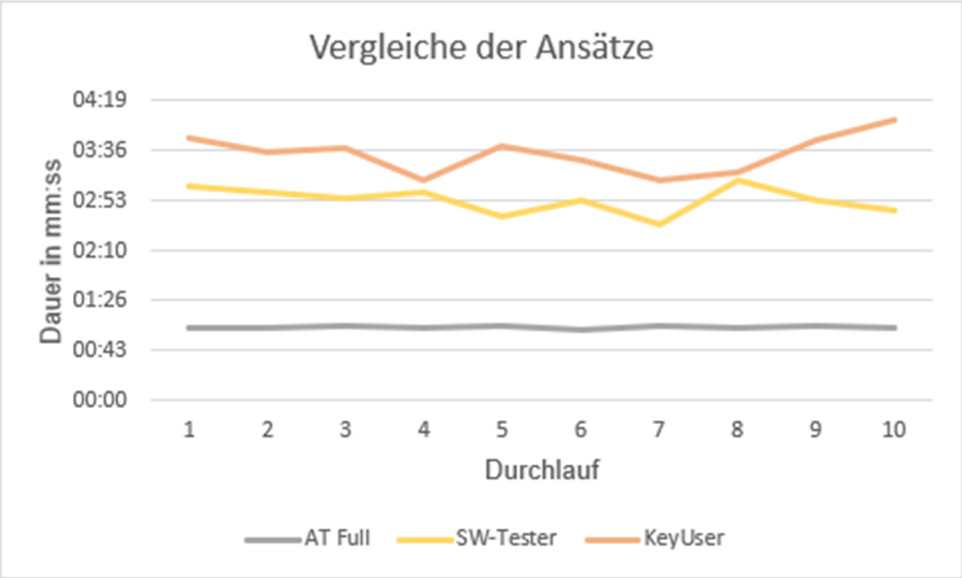

Abbildung 37: Zeitliche Vergleiche der Ansätze bei „Artikel kopieren“ (eigene Abbildung, 2023)

Auch der letzte überprüfte Modultest bietet im Gegensatz zu manuellen Tests eine Zeitersparnis. Während bei der Automatisierung der Artikel kopiert und überprüft wird, muss bei der manuellen Kopie nach der Artikelkopie ein weiteres Formular aufgerufen und dort die Werte verglichen werden.

5. ANALYSE DER ERGEBNISSE

In diesem Kapitel werden nun nochmals die Zeiten der in RSAT erstellten Testfälle und der programmierten Modultests mit den Testzeiten der KeyUser der Fachabteilungen verglichen, weiters werde ich näher auf die Erstellungsdauer der automatisierten Testfälle eingehen und erläutern, wo es bei der Testfallerstellung zu Komplikationen kam und wo man durch die bereitgestellten Werkzeuge limitiert war.

Am Ende des Kapitels folgt eine Zusammenfassung der gewonnenen Erkenntnisse und der Änderungen in der Testdurchführung.

5.1 Ergebnisse RSAT

Verglichen wurde die durchschnittliche Dauer der zehn durchgeführten Testdurchläufe der drei Testfälle „Artikelanlage“, „Rohstoff bestellen“ und „Stückliste erstellen“. Da die manuellen Tests in der Firma Wewalka Frischteig GmbH von den KeyUsers durchgeführt werden, sind deren benötigte Zeit der Vergleichswert für die Testautomatisierung. Tabelle 18 zeigt die Aufschlüsselung der benötigten Zeiten und deren Verhältnis zueinander.

	Artikelanlage (in mm:ss)	**Rohstoff bestellen (in mm:ss)**	**Stückliste erstellen (in mm:ss)**
Testautomatisierung	06:20	00:53	01:01
KeyUser	04:30	00:42	01:05
Automatisierung %	≈141%	≈126%	≈94%

Tabelle 18: Ergebnisse Vergleiche RSAT mit KeyUsers

Bei den überprüften Tests konnte die Testautomatisierung die Testdauer nur bei der „Stückliste erstellen“ verringern, hier wurden nur ≈94% der Zeit benötigt, die durchschnittlich von KeyUsers aufgewendet wird. Bei den anderen beiden waren die Testpersonen der Fachabteilungen schneller als die automatisierten Testfälle, RSAT benötigte hier ≈141% und ≈126% der manuellen Testzeit. Zu beachten ist aber, dass RSAT die Ergebnisse der durchgeführten Testfälle zentral in Azure DevOps speichert und so eine Historie und Details über die vergangenen Tests ablegt.

Die folgende Abbildung 38 zeigt grafisch noch einmal die Gegenüberstellung des Automatisierungs-Tools und der Testpersonen der Fachabteilungen.

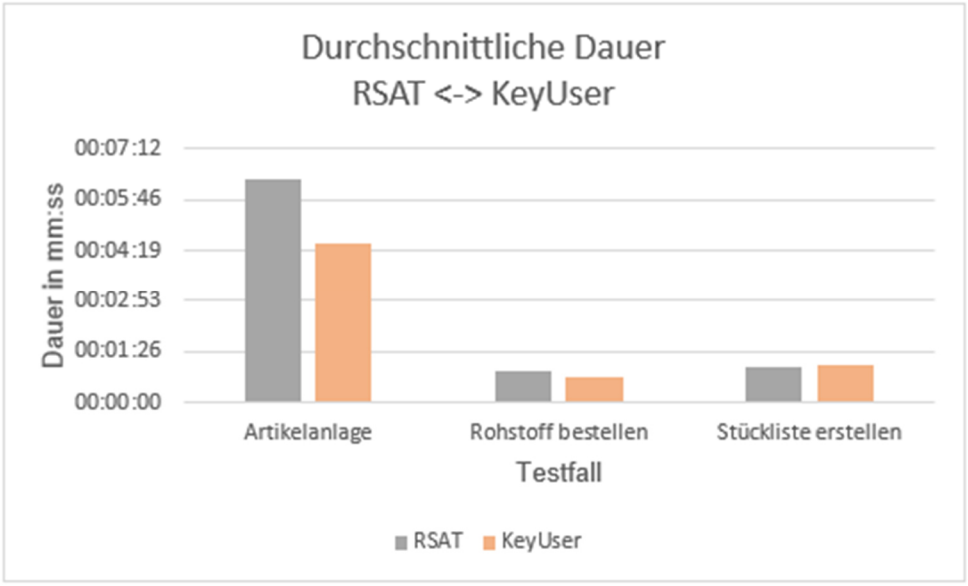

Abbildung 38: Durchschnittliche Dauer RSAT gegenüber KeyUsers (eigene Abbildung, 2023)

Jedoch müssen die Testfälle erst aufgenommen und angepasst werden. So wurde für die Erstellung und die Anpassung von „Artikelanlage“ acht Stunden von einem Programmierer gebraucht, die beiden anderen Testfälle benötigten jeweils drei Stunden Programmieraufwand. Auch wenn die manuellen Tests weniger Zeit benötigen als die automatisierten Durchläufe, die Arbeitszeit von Mitarbeiter*innen in den Fachabteilungen wird nicht mehr für die Durchführung von Tests in Beschlag genommen. Weiters werden Systemupdates und Installationen neuer Funktionen außerhalb der üblichen Arbeitszeit vorgenommen, so stören die automatisierten Tests den normalen Betrieb nicht.

5.2 Ergebnisse Modultests

Auch bei den Modultests wurde die durchschnittliche Dauer der zehn durchgeführten Testdurchläufe der drei Testfälle „Einheitenumrechnung“, „Mehrwertsteuer ermitteln“ und „Artikelkopie“ verglichen. Auch die bisherigen Tests, die durch Modultests ersetzt wurden, wurden von den KeyUsers durchgeführt, deshalb sind auch hier deren benötigte Zeit der Vergleichswert für die Testautomatisierung.

Tabelle 19 zeigt die Aufschlüsselung der benötigten Zeiten der drei Modultests und deren Verhältnis zueinander.

	Einheitenumrechnung (in mm:ss)	**Mehrwertsteuer ermitteln (in mm:ss)**	**Artikelkopie (in mm:ss)**
Testautomatisierung	01:05	01:03	01:03
KeyUser	62:23	05:23	03:34
Automatisierung %	≈2%	≈20%	≈30%

Tabelle 19: Ergebnisse Vergleiche Modultests mit KeyUsers

Bei den automatisierten Testfällen auf Code-Ebene konnte einer enorme Verbesserung der Testdauer erreicht werden. Da hier das Frontend und die Benutzeroberfläche des ERP-Systems nicht geladen und durchnavigiert werden muss und die Testfälle direkt mit den Datensätzen in der Datenbank arbeiten konnte die Dauer bei der „Einheitenumrechnung“ auf ≈2% der manuellen Dauer verringert werden. „Mehrwertsteuer ermitteln“ benötigt nur mehr ein Fünftel der manuellen Zeit und „Artikelkopie“ kann in etwa einem Drittel der manuellen Dauer überprüft werden. Zwar wird beim Build-Vorgang auch das Ergebnis der durchgeführten Testfälle angegeben, jedoch sind diese nicht so übersichtlich gestaltet wie die Dokumentation des RSAT. Hier ist der Programmierer gefordert, exakte und eindeutige Fehlerausgaben zu implementieren. Da die Modultests üblicherweise bei Programmierung einer entsprechenden Funktion erstellt werden fallen diese auch nicht sehr ins Gewicht. Alle drei Modultests wurden in etwa 10 Minuten entwickelt und überprüft. Abbildung 39 zeigt grafisch noch einmal die Gegenüberstellung der Modultests gegenüber den KeyUsers.

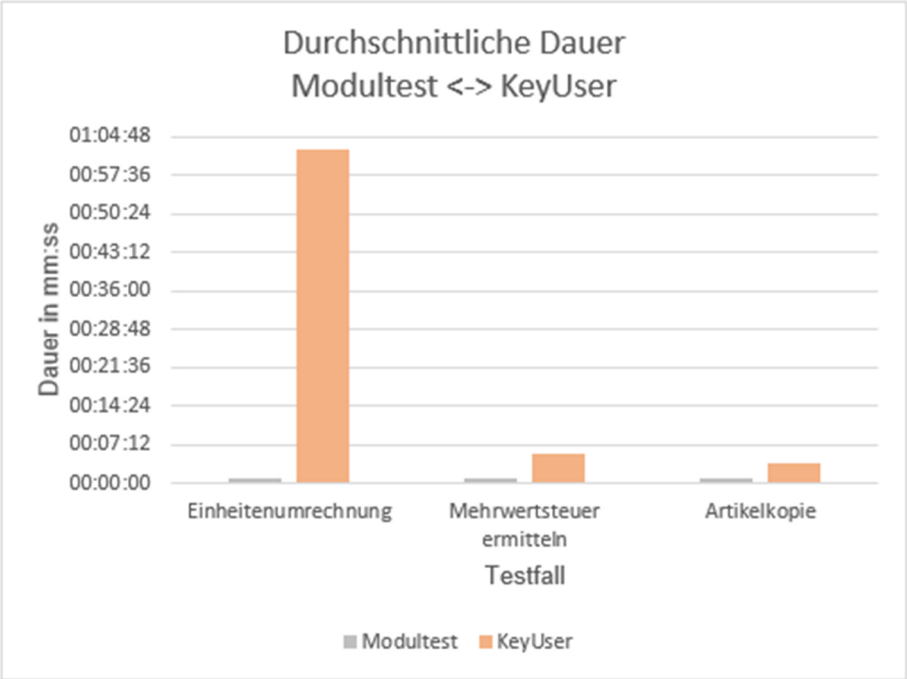

Abbildung 39: Durchschnittliche Dauer Modultests gegenüber KeyUsers (eigene Abbildung, 2023)

5.3 Verwendete Techniken

Um RSAT verwenden zu können müssen zuerst die Testfälle im ERP-System aufgezeichnet werden. Leider ist die technische Umsetzung des Werkzeugs nicht ausgereift. Zwar gibt es die Möglichkeit Testschritte im Nachhinein zwischen andere Testschritte einzufügen oder auch Testschritte zu löschen, jedoch bewirkt dies oft eine zerstörte Testaufzeichnung. Stabiler ist auf jeden Fall eine Testaufzeichnung ohne Fehlklicks oder Falscheingaben aufzuzeichnen, um ein nachträgliches Verändern zu vermeiden. Die Excel-Files mit den Testfallparametern sind zwar intuitiv gestaltet, jedoch ist oftmals ein „Trial-And-Error“-Prinzip bei benötigten Excel-Funktionen vonnöten, da situationsbedingt die wirklichen Werte, oder eben die Excel-Funktion übergeben werden muss.

Zusätzlich kann man mit der Aufgabenaufzeichnung nicht alle Funktionen des ERP-Systems nutzen. Manche Formulare können im System mit einem Rechtsklick geöffnet werden, diese Funktionalität ist aber im RSAT nicht implementiert. Hier muss man diese Funktionalität durch Umwege „simulieren“. Damit wird aber der in der Realität gelebte Prozess nicht getestet, sondern nur eine Abart davon.

Um die erstellten Testfälle mit dem RSAT zu testen, wird bei Wewalka Frischteig GmbH bei Systemupdates ein Bot-Programm auf der Plattform angeworfen, der die gewünschten Testfälle auf der aktualisierten Maschine durchläuft. Auch wenn diese Testfälle außerhalb der üblichen Arbeitszeiten stattfindet, ist man bei vielen Testfällen

und Durchläufen schnell bei einer Verdoppelung der Update-Dauer. So haben sich neue Fragen ergeben:

- Sollen alle Tests bei jedem Update durchgeführt werden oder sollen nur kritische Bereiche jedes Mal getestet werden?
- Können RSAT-Aufzeichnungen durch Modultests ersetzt werden?
- Kann das Update trotz fehlerhafter Testfälle eingespielt werden, die danach korrigiert werden?

Wir haben uns im Team bei der Frage, ob alle Tests jedes Mal durchgeführt werden, auf einen Kompromiss geeinigt. Es gibt kritische Bereiche, wie die Verladung und Freigabe der LKWs, deren Tests bei jedem Update durchgeführt werden. Bei systemunkritischen Bereichen wie Marketing werden die Tests nicht bei jedem Update durchgeführt, diese werden nur bei Updates durch Microsoft oder bei Veränderung der abteilungsspezifischen Tabellen und Formulare aktiviert.

Zwar wäre es möglich, den Großteil der RSAT-Aufzeichnungen durch Modultests zu ersetzen und die Testdauer zu verringern, jedoch geht uns dann auch die Überprüfung der Benutzeroberfläche verloren. Deswegen werden wir bei Tests, die eine Vielzahl an Formularfeldern besitzen, weiter RSAT mit der Aufgabenaufzeichnung verwenden.

Updates mit fehlgeschlagenen Testfällen werden trotzdem in das Stage-System eingespielt, die fehlerhaften Funktionen können dann in den nächsten Tagen korrigiert werden. So haben andere Abteilungen mit neuen, korrekten Funktionalitäten keine Einschränkungen durch ein fremdes, fehlerhaftes Modul.

5.4 Gewonnene Erkenntnisse

Durch den Einsatz der Testautomatisierung wird der zeitliche Aufwand der KeyUsers der Fachabteilungen verringert oder sogar komplett gestrichen, jedoch hat sich der gesamte Testaufwand selbst erhöht. So müssen bei Einsatz von RSAT die Testfälle aufgezeichnet und stabilisiert oder angepasst, zukünftig gegebenenfalls auch gewartet und die Testergebnisse kontrolliert werden. Auf der anderen Seite sorgen die Modultests für einen enormen Zeitgewinn, hier kann der zeitliche Testaufwand – abhängig vom entwickelten Modultest – um den Faktor 5 verringert werden, da die Programmierung der Modultests und deren Durchführung weit unter einer manuellen Überprüfung liegt. Ganzheitlich gesehen – und durch die Kombination von RSAT und Modultests – kann aber gesagt, werden, dass der gesamte zeitliche Testaufwand nicht um den Faktor 5 verringert werden kann.

Die hier gewonnenen Erkenntnisse und die Vergleiche der Testdauer lassen den Schluss zu, dass die erhobenen Ergebnisse im Gegensatz zu der im Eingangskapitel aufgestellten Hypothese steht.

Aber auch, wenn der Testaufwand erhöht wird, wird die Testautomatisierung in Zukunft die manuellen Testtätigkeiten zuerst unterstützen und letztendlich übernehmen.

Ein weiterer Vorteil ist, dass die automatisierten Tests zu jeder Zeit durchgeführt werden können. So wurde im Zuge der Erörterung der Testautomatisierung ein reiner Test-Server konfiguriert, der bei jedem Update oder Implementierung einer neuen Funktion auf den neuesten Stand gebracht wird und für gewünschte Testdurchführungen der Entwickler zur Verfügung steht. Dieser Test-Server wird auch verwendet, um neu erstellte automatisierte Tests zu testen. Da ein fehlerhafter oder unvollständiger automatisierter Test für korrupte Daten im Stage-System sorgen könnte, wird die Entwicklung und die Überprüfung der automatisierten Tests erst am Test-Server durchgeführt, bei stabilen Ergebnissen werden diese in das Stage-System übernommen und können von nun an in das Test-Portfolio übernommen werden.

6. SCHLUSSFOLGERUNGEN

Im letzten Kapitel werden die gewonnenen Erkenntnisse mit der Hypothese verglichen, die Forschungsfrage beantwortet und eine Zusammenfassung und Ausblick über die zukünftige Ausrichtung der Testautomatisierung der Firma Wewalka Frischteig GmbH gegeben.

6.1 Verifikation der Hypothese

Die im Eingangskapitel aufgestellte Hypothese lautet:

„Durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka wird der zeitliche Aufwand um den Faktor 5 verringert."

Zwar können die KeyUsers der Fachabteilungen bei den Testaufgaben unterstützt oder sogar von den Testaufgaben entbunden werden, jedoch erhöht sich der Aufwand bei den Programmierern enorm. Manuelle Testfälle müssen mittels Aufgabenaufzeichnung erstellt, angepasst und gewartet werden, neue Funktionen müssen mit dazugehörigen Modultests erstellt werden. Zusätzlich werden Anpassungen am Build-Prozess nötig, und eine geeignete Teststrategie für die Auswahl der Tests muss gefunden werden. Da auch bei Updates des Systems durch Microsoft Formulare und Tabellen geändert werden können, muss auch die Testfallwartung zum Testaufwand addiert werden.

Durch diese Tätigkeiten mit den im vorigen Kapitel erörterten Testfallzeiten kann die Hypothese nicht bestätigt werden.

6.2 Beantwortung der Forschungsfrage

Die Forschungsfrage dieser Arbeit lautet:

„Wird durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka der zeitliche Aufwand um den Faktor 5 verringert?"

Nach der Implementierung des ersten automatischen Test-Teilsystems und der Analyse der Ergebnisse lässt sich diese Forschungsfrage folgendermaßen beantworten: Der zeitliche Aufwand lässt sich durch den Einsatz von Automatisierung der Softwaretests nicht um den Faktor 5 verringern. Zwar können Modultests auf Code-Ebene die manuelle Testdauer immens verkürzen, jedoch werden vermehrt Entwickler-Ressourcen zur Erstellung, Anpassung und Wartung der automatisierten Tests benötigt. Ein großer Vorteil der Testautomatisierung ist die Tatsache, dass die Personen aus der Fachabteilungen auf komplexe Themen und auf das übrige Arbeitsumfeld konzentrieren können. Weiters laufen die automatisierten Testfälle außerhalb der üblichen Arbeitszeiten und sind vor menschlicher Übermüdung und „false positives" gefeit. Entwickler können auch von der zentralisierten Dokumentation profitieren, da eine

exakte Beschreibung und Ort des Fehlers vorliegen. Dies könnte zur schnelleren Fehlerfindung beitragen.

6.3 Zusammenfassung

In dieser Arbeit wurde die Forschungsfrage „Wird durch den Einsatz von Automatisierung der Softwaretests bei Änderungen des ERP-Systems der Firma Wewalka der zeitlich Aufwand um den Faktor 5 verringert?“ beantwortet. Zwar konnte der Testaufwand nicht um den Faktor 5 verringert, jedoch wurden durch Einsatz von Aufgabenaufzeichnung, RSAT und Modultests die Mitarbeiter des Unternehmens entlastet werden. Durch die Einbindung in die Build-Prozesse wird die Testautomatisierung außerhalb der üblichen Arbeitszeiten durchgeführt, die Gefahr menschlicher Übermüdung wird vermieden und „false positives“ ausgeschlossen.

Entwickler werden zu Beginn der Implementierung der Testautomatisierung höheren Aufwand zu stemmen haben und auch später für die Wartung der automatisierten Testfälle Ressourcen aufwenden müssen, jedoch profitieren auch diese durch eine zentrale Dokumentation der erfolgreichen und fehlgeschlagenen Durchläufe der Testfälle.

In dieser Arbeit wurde eine Primärdatenerhebung vorgenommen und drei Testfälle „Artikelanlage“, „Rohstoffe bestellen“ und „Stückliste erstellen“ mittels Aufgabenaufzeichnung erstellt und später per RSAT konfiguriert und abgespielt. Durch die Komplexität der „Artikelanlage“ wurde diese in eine Test Suite verfrachtet und ihre einzelnen Bereiche in Untertestfälle aufgesplittet. So ist es bei einem fehlerhaften Testfall einfacher, die schadhafte Stelle auf einen Blick zu erkennen. Zusätzlich wurden drei Modultests angelegt, die die Funktionen „Einheitenumrechnung“, „Mehrwertsteuer ermitteln“ und „Artikelkopie“ überprüften. Bei Modultests „Einheitenumrechnung“ ist die Stärke der Testautomatisierung deutlich sichtbar. Während auf der Benutzeroberfläche verschiedene Masken aufgerufen werden müssen, könne diese auf Datenbankebene miteinander verknüpft werden und benötigen nur ~2% der manuellen Testdauer, wobei zusätzlich auch noch Flüchtigkeitsfehler und Übermüdung vermieden werden können.

6.4 Ausblick

Das derzeit eingesetzte Teilsystem des automatisierten Software-Tests wurde bisher positiv aufgenommen. Die Mitarbeiter der Fachabteilungen sind zufrieden, nicht mehr für Tests im System aus ihrem Tagesgeschäft gerissen zu werden. Durch die zentrale Dokumentation sind Fehlerfälle auch einfach und in kürzester Zeit zu finden. Modultests werden in Zukunft bei Testfällen zum Einsatz kommen, die wenig Interaktion mit Tabellen und Formularen im ERP-System beinhalten.

Durch die Einschränkung des RSAT wird gerade eine Alternative überprüft. Mit dem Selenium Webdriver und der Programmiersprache Python soll das browserbasierte ERP-System durchgetestet werden. Anders als beim RSAT ist man in der Programmierung frei und kann alle Funktionen programmieren, die die Programmiersprache zulässt. So konnte bereits ein Testfall erstellt werden, der einen Bericht im PDF-Format speichert, diese Werte ausliest und in einem CSV-File speichert. So können zukünftig auch verschiedenste Dateiformate, die das ERP-System anbietet auf korrekte Werte, oder auch korrekte Positionierung überprüft werden.

Unabhängig, welche der verfügbaren Automatisierungsmöglichkeiten dann zum Einsatz kommen, werden die manuellen Testtätigkeiten in Zukunft sukzessive von der Testautomatisierung abgelöst werden.

Im April 2023 gab es ein verpflichtendes Plattform-Update von Microsoft, bei dem Module aktiviert wurden, die für eine gewisse Konstellation eine Bereitstellung von Waren und Erstellung von Ladungen für LKWs nahezu unmöglich machten. Leider wurde genau dieser Prozess nicht manuell und automatisiert überprüft, da es in dieser speziellen Funktion seit Jahren keine Änderung gab. Mit einer Testautomatisierung wäre dieser Fehler sofort aufgefallen, leider wurde er erst im Produktiv-System bei einer echten Warenbereitstellung entdeckt. Dieser und ähnliche Fälle dürften in Zukunft der Vergangenheit angehören, da eine vollständige Systemüberprüfung bei Plattform-Updates geplant ist.

Literaturverzeichnis

Beck, Kent, Ward Cunningham, Ron Jeffries. 2001. *Agile Manifesto*. Abgerufen am 04. Dezember 2022. http://agilemanifesto.org

Beck, Kent. 2002. *Test Driven Development: By Example*. Boston: Addison-Wesley https://dl.acm.org/doi/10.5555/579193

Bucsics, Thomas, Manfred Baumgartner, Richard Seidl, Stefan Gwihs. 2015. „Standards und Normen“. In *Basiswissen Testautomatisierung - Konzepte, Methoden und Techniken*, 2-4. Heidelberg: dpunkt-Verlag

Dustin, Elfriede, Jeff Rashka und John Paul. 1999. „The Automated Test Life-Cycle Methodology (ATLM).” In *Automated Software Testing: Introduction, Management, and Performance*, 7-14. Boston: Addison-Wesley

Dustin, Elfriede, Jeff Rashka und John Paul. 2001. “Entstehung und Entwicklung des automatisierten Testens.“ In *Software automatisch testen*, 3-14. Berlin, Heidelberg: Xpert.press. Springer. https://doi.org/10.1007/978-3-642-56806-0

Duvall, Paul, Stephen Matyas und Andrew Glover. 2007. “A Day in the Life of CI”. In *Continuous Integration: Improving Software Quality and Reducing Risk*, 25-29. Boston: Addison-Wesley. EPUB.

Ebert, Christof. 2011. *Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing*. Wiley-IEEE Computer Society Pr. 1st edition (November 8, 2011) https://www.wiley.com/en-us/Global+Software+and+IT%3A+A+Guide+to+Distributed+Development%2C+Projects%2C+and+Outsourcing-p-9780470636190

Fowler, Martin, Matthew Foemmel. 2006.*Continuous Integration*. 1. Mai. Abgerufen am 18. September 2022. https://www.martinfowler.com/articles/continuousIntegration.html

ISTQB. 2015. International Software Testing Qualifications Board. Abgerufen am 27. Februar 2023. https://istqb-glossary.page/de/sylabi/foundation-extension-model-based-testing-2015/

Palamarchuk, Sofia. 2015. *The True ROI of Test Automation*, Abstracta. Abgerufen am 06. Jänner 2023. https://abstracta.us/blog/test-automation/true-roi-test-automation/

Pham, Phuoc, Vu Nguyen, Tien Nguyen. 2022. *A Review of AI-augmented End-to-End Test Automation Tools.* Association for Computing Machinery. https://dl.acm.org/doi/abs/10.1145/3551349.3563240

Pol, Martin, Tim Kommen, Andreas Spillner. 2002. *Management und Optimierung des Testprozesses.* Heidelberg: dpunkt-Verlag

Polo, Macario, Pedro Reales Mateo, Mario Piattine, Christof Ebert. 2013. *Test Automation.* IEEE Software. Abgerufen am 07. Januar 2023. https://doi.org/10.1109/MS.2013.15

Wang, Yuqing, Mika V. Mäntylä, Zihao Liu, Jouni Markkula, Päivi Raulamo-Jurvanen. 2022. Improving test automation maturity: A multivocal literature review, Journal of Software: Testing, Verification and Reliability, Volume 32, Issue 3, Wiley. Abgerufen am 11. Januar 2023. https://doi.org/10.1002/stvr.1804

Wiklund, Kristian, Sigrid Eldh, Daniel Sundmark, Kristina Lundqvist. 2017. Impediments for software test automation: A systematic literature review, Journal of Software: Testing, Verification and Reliability, Volume 27, Issue 8, Wiley. Abgerufen am 11. Januar 2023. https://doi.org/10.1002/stvr.1639

Abbildungsverzeichnis

Abbildung 1: Abgedeckte Teilbereiche des Software-Tests. (eigene Abbildung in Anlehnung an Dustin et al. 2001) 6

Abbildung 2: Continuous Integration mit Build Process. (eigene Abbildung mit Anlehnung an Duvall et al, 2007) 10

Abbildung 3: Deployment und Test bei Wewalka (eigene Abbildung, 2022) 11

Abbildung 4: Test in jeder Iteration der agilen Software-Entwicklung. (eigene Abbildung, 2022).....18

Abbildung 5: Verlauf der Kostenkurven bei Ausführung von manuellen und automatischen Tests (Palamarchuk, 2015) 19

Abbildung 6: Red-Green-Factoring (eigene Abbildung, 2022) 20

Abbildung 7: Hürden bei der Einführung von Testautomatisierung. (eigene Abbildung in Anlehnung an Wiklund et al., 2017).....22

Abbildung 8: Auswahl Testautomatisierung AI/ML (eigene Abbildung, 2023) 27

Abbildung 9: Kategorisierung der Testfälle (eigene Abbildung, 2023).....29

Abbildung 10: Erstellen einer Aufzeichnung (eigene Abbildung, 2023).....32

Abbildung 11: Fertige Aufgabenaufzeichnung (eigene Abbildung, 2023).....32

Abbildung 12: Filter von Artikelnummer (eigene Abbildung, 2023).....34

Abbildung 13: Filter von Artikelgruppe (eigene Abbildung, 2023).....34

Abbildung 14: Nachricht/Warnung von Microsofts D365 Finance & Operations (eigene Abbildung, 2023).....35

Abbildung 15: Custom Parameters (eigene Abbilung, 2023) 40

Abbildung 16: Fehlermeldung bei falscher Löschreihenfolge (eigene Abbildung, 2023) 43

Abbildung 17: Dauer Testfall / Tool-Init. (eigene Abbildung, 2023) 49

Abbildung 18: Zeitliche Vergleiche der Ansätze bei „Artikel anlegen“ (eigene Abbildung, 2023).....50

Abbildung 19: Zeitliche Vergleiche der Ansätze bei „Rohstoffbestellung“ (eigene Abbildung, 2023).....53

Abbildung 20: Vereinfachte Stückliste eines Pizzateiges (eigene Abbildung, 2023).....53

Abbildung 21: Zeitliche Vergleiche der Ansätze bei „Stückliste erstellen“ (eigene Abbildung, 2023).....56

Abbildung 22: Dokumentierte Ergebnisse Test Suite "Artikelanlage" (eigene Abbildung, 2023)....57
Abbildung 23: Fehler bei "Artikel anlegen" (eigene Abbildung, 2023)....58
Abbildung 24: Error Log in Azure DevOps (eigene Abbildung, 2023)....58
Abbildung 25: Fehler bei "Artikel prüfen" (eigene Abbildung, 2023)....59
Abbildung 26: Error Log in Azure DevOps (eigene Abbildung, 2023)....60
Abbildung 27: Diagramme von "Artikelanlage" (eigene Abbildung, 2023)....61
Abbildung 28: Fehlgeschlagene Umwandlung von "Karton" zu "Stück" (eigene Abbildung, 2023)....63
Abbildung 29: Fehlgeschlagene Umwandlung von "Palette" zu "Kartons" (eigene Abbildung, 2023)....63
Abbildung 30: Fehlgeschlagene "FREI"-Überprüfung (eigene Abbildung, 2023)....65
Abbildung 31: Fehlgeschlagene "E20"-Überprüfung (eigene Abbildung, 2023)....65
Abbildung 32: Fehlerhafte, doppelte Artikelnummer (eigene Abbildung, 2023)....67
Abbildung 33: Fehlerhafte Tiefe bei der Artikelkopie (eigene Abbildung, 2023)....67
Abbildung 34: Dauer der Modultests (eigene Abbildung, 2023)....68
Abbildung 35: Zeitliche Vergleiche der Ansätze bei „Einheitenumrechnung“ (eigene Abbildung, 2023)....69
Abbildung 36: Zeitliche Vergleiche der Ansätze bei „Mehrwertsteuer ermitteln“ (eigene Abbildung, 2023)....70
Abbildung 37: Zeitliche Vergleiche der Ansätze bei „Artikel kopieren“ (eigene Abbildung, 2023)....71
Abbildung 38: Durchschnittliche Dauer RSAT gegenüber KeyUsers (eigene Abbildung, 2023)....73
Abbildung 39: Durchschnittliche Dauer Modultests gegenüber KeyUsers (eigene Abbildung, 2023)....75

Tabellenverzeichnis

Tabelle 1: Parameter des Testfalls 34
Tabelle 2: Testschritte „Artikel prüfen“ 36
Tabelle 3: Gespeicherte Variablen des Testfalls 37
Tabelle 4: Testschritte „Artikel anlegen" 39
Tabelle 5: Testschritte "Artikel prüfen" 42
Tabelle 6: Testschritte "Artikel löschen" 43
Tabelle 7: Testschritte "Produkt löschen" 44
Tabelle 8: Testreihenfolge 45
Tabelle 9: Zeiten der zehn durchgeführten Durchläufe „Artikel anlegen“ 48
Tabelle 10: Anteil der Werkzeug-Initialisierung an der Gesamtzeit 49
Tabelle 11: Testfall "Rohstoff bestellen" 52
Tabelle 12: Zeiten der zehn durchgeführten Durchläufe "Rohstoff bestellen" 52
Tabelle 13: Testfall "Stückliste erstellen" 55
Tabelle 14: Zeiten der zehn durchgeführten Durchläufe "Stückliste erstellen" 56
Tabelle 15: Zeiten der zehn durchgeführten Durchläufe "Einheitenumrechnung" 69
Tabelle 16: Zeiten der zehn durchgeführten Durchläufe "Mehrwertsteuer ermitteln" 70
Tabelle 17: Zeiten der zehn durchgeführten Durchläufe "Artikel kopieren" 71
Tabelle 18: Ergebnisse Vergleiche RSAT mit KeyUsers 72
Tabelle 19: Ergebnisse Vergleiche Modultests mit KeyUsers 74

Abkürzungsverzeichnis

AI …… Artificial Intelligence

API ……. Application Programming Interface

AT …….. Automatischer Test / Automated Test

BLT …… Blätterteig (Wewalka-spezifisch)

CD …….. Continuous Delivery

CI …… Continuous Integration

CLI ……. Command Line Interface

CSV …… Comma-Separated-Values

ERP …… Enterprise-Resource-Planning

ETSI ……. European Telecommunication Standards Institute

GUI ……. Graphical User Interface

KI …… Künstliche Intelligenz

MHD …….. Mindesthaltbarkeitsdatum

ML …….. Machine Learning

PZT …… Pizzateig

ROI ……. Return On Investment

RSAT ……. Regression Suite Automation Tool

TDD …… Test Driven Development

TTCN-3 ……. Testing and Test Control Notation

UI …… User Interface

USP …… Unique Selling Proposition (Alleinstellungsmerkmal)

VCS …… Version Control System