Entwicklung einer integrierten Microservice-Architektur am Beispiel von modularisierten RPA-Prozessen

Max-Arthur Klink

Hochschule Pforzheim
Tiefenbronnerstr. 65
75175 Pforzheim
klinkmax@hs-pforzheim.de

Frank Morelli

Hochschule Pforzheim
Tiefenbronner Straße 65
75175 Pforzheim
frank.morelli@hs-pforzheim.de

ABSTRACT
Die vorliegende Ausarbeitung untersucht die Entwicklung einer modularen Microservice-Architektur zur Optimierung von Robotic Process Automation (RPA). Ziel ist es, Flexibilität, Skalierbarkeit und Wartbarkeit zu verbessern, indem monolithische RPA-Prozesse in unabhängige, wiederverwendbare Microservices aufgeteilt werden. Ein praxisnahes Implementierungsmodell adressiert dabei zentrale Anforderungen wie Modularität, lose Kopplung und Resilienz. Die Differenzierung zwischen wertgenerierenden und unterstützenden Microservices ermöglicht eine effiziente Prozessgestaltung, während ein zentraler Katalog und Orchestrierungswerkzeuge die Verwaltung und Integration erleichtern. Experteninterviews lieferten fundierte Einblicke in die Priorisierung relevanter Architekturmerkmale. Die Ergebnisse zeigen, dass die entwickelte Architektur wesentliche Effizienzgewinne und eine erhöhte Anpassungsfähigkeit ermöglicht. Abschließend wird die Architektur hinsichtlich zentraler Mehrwerte evaluiert, und es werden konkrete Handlungsempfehlungen für zukünftige Anwendungen und Erweiterungen gegeben.

SCHLÜSSELWÖRTER
Microservices; Robotic Process Automation (RPA); Architektur; Modularisierung; Prozessautomatisierung; Architekturprinzipien; Implementierung.

EINLEITUNG

Die Automatisierung von Geschäftsprozessen durch den Einsatz von Robotic Process Automation (RPA) ist ein zentraler und stetig wachsender Bereich in der IT-Strategie vieler Unternehmen. Durch die Verwendung dieser Technologie können standardisierbare Prozesse, welche in der Komplexität variieren, automatisiert werden. Aktuell erfolgt die Analyse und Automatisierung von Geschäftsprozessen jeweils getrennt voneinander. Allerdings treten bei einer Vielzahl von Geschäftsprozessen Ähnlichkeiten im Hinblick auf Inhalte und Struktur der darunter liegenden Teilprozesse auf. Dies führt häufig dazu, dass man innerhalb dieser Prozesse ähnliche Funktionen bzw. gleiche Bestandteile verwendet. Die Automatisierung von Geschäftsprozessen verfolgt in diesem Fall einen monolithischen Ansatz. Dies bedeutet, dass die gesamte Logik und Funktionalität innerhalb jedes einzelnen automatisierten Prozesses integriert ist.
Die monolithische Struktur der aktuellen RPA-Prozesse bringt verschiedene Herausforderungen mit sich, insbesondere hinsichtlich der Prozesswiederverwendbarkeit und -wartbarkeit. Steigende oder sich ändernde Anforderungen an die Automatisierung erfordern oft Anpassungen mehrere Komponenten innerhalb eines Prozesses, wodurch die Aktualisierung komplex und zeitaufwändig ist. Änderungen an einzelnen Bestandteilen können mehrere Prozesse betreffen, wodurch diese ebenfalls geändert werden müssen, was zusätzlich die Wartbarkeit und Flexibilität der Prozesse einschränkt. Dies zeigt den Bedarf nach einer effizienteren Lösung für die Konzipierung von RPA-Prozessen auf.
Durch die Unterteillung der einzelnen Bestandteile der Prozesse in unabhängige Microservices lassen sich zukünftig Funktionen und Komponenten separat entwickeln und bereitstellen. Damit soll eine einfachere Wartung und Aktualisierung der gesamten Prozesslandschaft erzielt werden, da man Änderungen nur noch in den betroffenen Microservices vornehmen muss und nicht in jedem einzelnen automatisierten Prozess. Die Zielsetzung des vorliegenden Artikels besteht in der Veranschaulichung einer Microservice-Architektur am Beispiel von modularisierten RPA-Prozessen. Der vorliegende Artikel untersucht die Thematik aus folgender Forschungsperspektive:

1. Wie lassen sich Microservices katalogisieren?
2. Wie kann eine Microservice-Architektur im RPA-Umfeld ausgestaltet werden?

ROBOTIC PROCESS AUTOMATION

Definition und Grundlagen

Bei Robotic Process Automation handelt es sich um eine Art der Prozessautomatisierung. Unter der Prozessautomatisierung wird der Einsatz von Software und Technologien zur Automatisierung von Geschäftsprozessen verstanden. Das Ziel der Prozessautomatisierung ist die Erreichung der Geschäftsziele, die Verbesserung der Rentabilität und der Wettbewerbsfähigkeit der Unternehmen (SAP o. D.). Der Begriff RPA kam erstmals im Jahr 2000 auf, fand jedoch bis zum Jahr 2012 kaum Verwendung und erlangte erst dann an Bedeutung (Doguc 2020, S. 470 f.). Im Herbst 2015 befand sich RPA in der Phase der frühen Mehrheit, was bedeutet, dass die Technologie bereits von einer Vielzahl von Unternehmen eingesetzt wird

(Willcocks et al. 2015, S. 3; Karnowski 2013, S. 520). Mit dem Verlauf der Jahre konnte ein signifikantes Wachstum in diesem Bereich beobachtet werden (Doguc 2020, S. 471). Es gibt eine Vielzahl an Definitionen für RPA. Der vorliegende Artikel basiert auf folgender Definition: „Bei RPA handelt es sich um eine Technologie, die es Unternehmen ermöglicht, repetitive, zeitaufwändige und regelbasierte Aufgaben zu automatisieren, wodurch menschliche Mitarbeiter entlastet und die Effizienz gesteigert wird." RPA trägt so zur Verbesserung der Geschäftsprozesse und zur Erreichung strategischer Unternehmensziele bei. Bei RPA handelt es sich um ein Software-Programm mit dem Softwareroboter programmiert werden können (Institute for Robotic Process Automation & Artificial Intelligence o. D.; Gartner o. D.b; PWC South Africa o. D.; Langmann und Turi 2021, S. 6). Diese Softwareroboter interagieren dabei mit der Präsentationsschicht anderer Programme. Sie verhalten sich dabei wie ein Mensch gegenüber der grafischen Benutzeroberfläche des Systems, ohne andere Schichten zu verwenden (Willcocks et al. 2015, S. 7 f.; van der Aalst et al. 2018, S. 269). Moderne RPA-Lösungen erweitern zusätzlich ihren Anwendungsbereich, indem sie neben der grafischen Oberflächenautomatisierung auch die Integration von API-Aufrufen ermöglichen. Durch die Verknüpfung mit Backend-Systemen über APIs lassen sich sowohl Front-End- als auch Back-End-Automatisierungen realisieren. Eine Studie hat zudem gezeigt, dass die Verwendung von APIs im RPA-Kontext gegenüber der reinen GUI-Automatisierung zu empfehlen ist, da unter anderem die Ausführungsgeschwindigkeit gesteigert werden kann (Průcha und Skrbek 2022, S. 260-262, 264, 267-272; AWS o. D.).

Der Abbildung 1 kann das RPA-Schichtmodell entnommen werden. Entsprechend der in der Literatur dargestellten Ansichten operiert RPA im Gegensatz zu anderen Automatisierungslösungen prinzipiell auf der Präsentationsschicht. Es ist jedoch anzumerken, dass insbesondere neuere RPA-Systeme erweiterte Funktionen bieten, die auch die Interaktion mit anderen Schichten, wie der Geschäftslogik- und Datenebene, ermöglichen (Drawehn et al. 2022, S. 14).

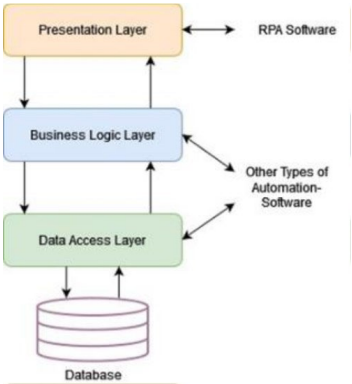

Abbildung 1: RPA-Schichtmodell Quelle: Eigene Darstellung in Anlehnung an Dey und Das 2019, S. 222; Drawehn et al. 2022, S. 14

Indem die RPA-Software auf den bestehenden Systemen aufsetzt, kann es auf der vorhandenen Infrastruktur implementiert werden, ohne dass Änderungen am IT-Backend erforderlich sind (Willcocks et al. 2015, S. 7; Berruti et al. 2017). Daraus folgt, dass RPA auf einen sogenannten „outside-in" Ansatz setzt, da das Informationssystem unverändert bleibt. In einem klassischen „inside-out" Ansatz, wären Änderungen am Informationssystem die Folge (van der Aalst et al. 2018, S. 269, 271). Weiterhin lässt sich RPA dem "Lightweight IT"-Konzept zuordnen: Hierunter fallen IT-Systeme, die unkompliziert auf die Bedürfnisse erfahrener Nutzer reagieren, ohne auf komplexe und umfassende IT-Infrastrukturen angewiesen zu sein (Bygstad 2017, S. 182; Willcocks et al. 2015, S. 21 f.). Innerhalb von RPA kann man zwischen drei Typen unterscheiden: Attended, Unattended und Hybrides RPA (Axmann und Harmoko 2020, S. 559).

Bei Attended RPA, auch als Robotic Desktop Automation (RDA) bezeichnet, kann ein Software-Roboter direkt auf dem Desktop des Benutzers ausgeführt werden. Der Benutzer ist in der Lage den Roboter zu starten, zu überwachen und mit ihm über einen Bildschirm zu interagieren. Der Roboter seinerseits kann mit verschiedenen Anwendungen interagieren wodurch sich verschiedene Arbeitsschritte automatisieren lassen. Attended RPA fungiert wie ein persönlicher Assistent, weil es bestimmte Aufgaben übernimmt und ausführt. Ein Nachteil von Attended RPA besteht darin, dass der Roboter den Computer des Users benötigt. Während der Programmlaufzeit ist dieser dann nicht mehr in der Lage, seinen Computer anderweitig zu verwenden (Langmann und Turi 2021, S. 6; Axmann und Harmoko 2020, S. 559). Eine Untersuchung von Anwendungsszenarien ergibt, dass Atended RPA primär bei Prozessen zum Einsatz kommt, die nicht komplett regelbasiert automatisierbar sind bzw. an verschiedenen Stellen menschliche Entscheidungen benötigen. Ein Anwendungsfall für Attended RPA ist z.B. die Wirtschaftsprüfung. Innerhalb dieser sind viele Prozesse unstrukturiert und kommen deshalb nicht ohne menschliche Interaktion aus (Zhang et al. 2021, S. 5, 7 f.).

Im Gegensatz zu Attended RPA bezeichnet Unattended RPA einen RPA-Typen, bei dem die Software Roboter statt auf dem Desktop des Benutzers, auf einem Server bzw. auf einer virtuellen Maschine im Hintergrund ausgeführt werden. Sie können unabhängig von Menschen arbeiten und benötigen meistens keine direkte Interaktion mit diesen. Durch ihre Unabhängigkeit lassen sie sich auf Basis von einer vordefinierten Uhrzeit oder eines festgelegten Triggers, wie der Erhalt einer E-Mail, automatisch triggern. Unattended RPA-Roboter werden mithilfe eines Orchestrators gesteuert und überwacht, eine Schlüsselkomponente von RPA-Systemen. Beleuchtet man die Anwendungsfälle für Unattended RPA näher, so eignet sich dieser RPA-Typ besonders gut für regelbasierte Anwendungsfälle. So lassen sich Anwendungsfälle lassen sich im Rechnungswesen & Controlling Bereich finden, wenn z.B. per E-Mail erhaltene Rechnungen automatisch nach gewissen Regeln verbucht werden sollen (Zhang et al. 2021, S. 5, 7 f.; Langmann und Turi 2021, S. 6 f.; Axmann und Harmoko 2020, S. 559;

Choi et al. 2021, S. 3 f.)
Bei Hybridem RPA handelt es sich um eine Kombination von Attended und Unattended RPA. Es eignet sich vorwiegend für komplexe Prozesse, bei denen ein Teil vollautomatisiert ohne menschliche Interaktion abläuft, während der andere Teil des Prozesses auf die Interaktion mit einem Menschen angewiesen ist (Axmann und Harmoko 2020, S. 559 f.).
Generell besteht ein RPA-System i.d.R. aus drei Hauptkomponenten, dem RPA Studio, dem RPA Orchestrator und den RPA-Robotern. Der Aufbau eines solchen Systems entnommen werden. Das RPA Studio repräsentiert die Entwicklungsumgebung von RPA. Dort lassen sich die Prozesse, welche in Form eines Bots ausgeführt werden sollen, modellieren/ entwickeln und konfigurieren werden. Ist ein Bot fertig entwickelt, wird er dem Orchestrator übergeben. Der Orchestrator dient als zentrale Steuerungseinheit für die Verwaltung der Bots. Konkret ist er für die Planung, die Ausführung und das Monitoring der Bots zuständig. Der Orchestrator bietet in der Regel auch eine Schnittstelle, über die Anwendungen von Drittanbietern die RPA-Roboter nutzen können. Die Roboter führen dann die ihnen zugewiesenen Aufgaben aus (Choi et al. 2021, S. 4).

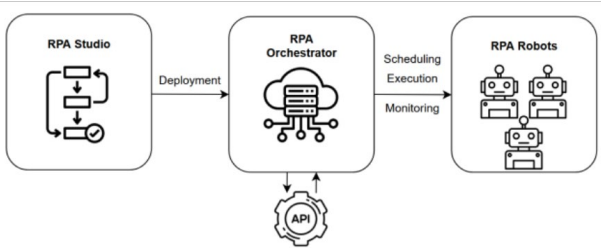

Abbildung 2 RPA-System Aufbau Quelle: Eigene Darstellung in Anlehnung an Choi et al. 2021, S. 4

Einsatzbereiche
RPA findet in einer Vielzahl von Bereichen Verwendung. Generell ist der Einsatz von RPA besonders geeignet für die Automatisierung von Geschäftsprozessen, die sich durch Regelmäßigkeit und Routinetätigkeiten auszeichnen. Weiter ist RPA geeignet für Prozesse, die strukturierte Daten verarbeiten, durch ein hohes Volumen geprägt sind und eine Interaktion mit mehreren IT-Systemen über die Oberfläche erfordern. RPA ist somit besonders für Prozesse geeignet, die keine Kreativität oder Interpretation erfordern (Da Costa et al. 2022, S. 8; Aguirre und Rodriguez 2017, S. 65 f., 70).
RPA findet in vielen unterschiedlichen Branchen Anwendung. Besonders häufig wird RPA in Bereichen wie IT, Personalwesen/HR, Versicherung, Buchhaltung und Finanzen, Einzelhandel sowie in der Wirtschaftsprüfung eingesetzt (Santos et al. 2020, S. 406; Kokina und Blanchette 2019, S. 1; Moffitt et al. 2018, S. 1, 9; Zhang et al. 2021, S. 2; Madakam et al. 2019, S. 1, 13). Die Aufgaben, die RPA dabei ausführt, sind oft das systematische Erfassen und Überprüfen von Daten, das Bearbeiten und Umstrukturieren von Dateien, das Anpassen von Formaten sowie das Synchronisieren und Abgleichen von Daten über mehrere Plattformen hinweg (Alberth und Mattern 2017, S. 58). Die aufgeführten Einsatzbereiche zeigen, dass RPA bereits eine tragende Rolle in der Optimierung und Automatisierung von Geschäftsprozessen einnimmt. Die Weiterentwicklung der RPA-Funktionalität erfolgt laufend, was unter anderem durch die Innovationen und Entwicklungen innerhalb des Informationstechnologiesektors vorangetrieben wird. Als Aktuelle Trends lassen sich folgende Trends ermitteln: Artificial Intelligence (AI) bzw. künstliche Intelligenz (KI), vermag die Effizienz zu steigern, indem die Integration von KI-Technologien RPA autonomer und dynamischer ausgestaltet. RPA kann hierdurch selbstständiger auf unterschiedliche Situationen reagieren und Abläufe kombinieren. Ein Beispiel hierfür ist die Verbesserung der Interaktion mit menschlichen Benutzern, indem empfangene Nachrichten korrekt interpretiert und automatisch beantwortet werden (Hanussek 2019). Weiter kann KI eine Verarbeitung von unstrukturierten Daten die Nutzung von Spracherkennung, die Verwendung von maschinellem Lernen und neuronalen Netzen im RPA Kontext ermöglichen. Neuronale Netze sind in der Lage, komplexe Muster in Daten zu erkennen, wodurch sich anspruchsvollere Aufgaben automatisieren lassen (UiPath o. D.; Köhler-Schute 2020, S. 29). Ein weiterer aktueller Trend ist die Verbindung von Process Mining mit RPA. Die Verwendung von Process Mining Software, wie z. B. Celonis, kann es Unternehmen ermöglichen, Prozesse leichter und effizienter zu identifizieren, die für eine Automatisierung im Kontext von RPA geeignet sind (Choi et al. 2022, S. 39604, 39611; Celonis o. D.). Ein weiterer aktueller Trend ist die Untersuchung des Einsatzes von Blockchain Technologien, z.B. bei Kryptowährungen wie Bitcoin oder Ethereum. Die Verbindung zu RPA ermöglicht es, Sicherheits- und Audit-Herausforderungen zu adressieren (Al-Slais und Ali 2023).

Vorteile und Herausforderungen
RPA hat sich als eine bedeutende Technologie innerhalb der digitalen Transformation etabliert. Jedoch besitzt RPA neben zahlreichen Vorteilen auch einige Herausforderungen, die Unternehmen bewältigen müssen, um bestmöglich von RPA zu profitieren.
Einer der wichtigsten Vorteile ist die erhebliche Effizienzsteigerung, die Unternehmen durch die Nutzung von RPA erreichen können. In einer Fallstudie wurde analysiert, wie effizient eine Gruppe unter Verwendung von RPA im Vergleich zu einer Gruppe ohne den Einsatz von RPA arbeitet. In dem Team, welche RPA-Software verwendet hat, konnten 21 % mehr Fälle bearbeitet werden als in der Vergleichsgruppe. Entsprechend eröffnet dies Wertsteigerungspotenziale, indem Mitarbeiter die durch Automatisierung gewonnene Zeit für wichtigere und komplexere Aufgaben nutzen können. Im Rahmen der genannten Fallstudie ist allerdings zu ergänzen, dass die Gruppe mit RPA nicht signifikant schneller waren

als die ohne RPA (Aguirre und Rodriguez 2017, S. 68-70; Shidaganti et al. 2021, S. 1). Ein weiterer Vorteil von RPA für Unternehmen sind die damit verbundenen Kosteneinsparungen, welche unter anderem durch die Automatisierung der Geschäftsprozesse entstehen. Diese Kosteneinsparungen ergeben sich zum einen daraus, dass man Mitarbeiter gezielter und effizienter einsetzen kann aufgrund der Automatisierung von Routinetätigkeiten. Zusätzliche Einsparungen ergeben sich dadurch, dass RPA die Fehlerquote bei Aufgaben wie der Dateneingabe signifikant reduzieren kann. Hierdurch sinken die Fehlerbehebungskosten und zeitgleich steigt die Qualität der auszuführenden Arbeit (Ivančić et al. 2019, S. 280, 282, 287 ,290; Kirchmer 2017, S. 2 f.; Asatiani und Penttinen Esko 2016, S. 4). Zusätzlich kann ein Roboter im Gegensatz zu einem Mitarbeiter rund um die Uhr (24/7) arbeiten. Eine Studie aus dem Jahr 2016 zeigt auf, dass die Kosten für einen RPA-Roboter nur ein Drittel bis ein Fünftel der Kosten eines Vollzeitmitarbeiters betragen können (Kroll et al. 2016, S. 12; Asatiani und Penttinen Esko 2016, S. 4). Konkret können Unternehmen mit einer durchschnittlichen Kosteneinsparung von ca. 25 % rechnen. Der Break-Even-Point von RPA wird dabei meistens schon im ersten Jahr nach der Einführung erreicht, mit einem potenziellen Return on Investment (ROI) von 30 bis 200 % (Koch und Wildner 2020, S. 214; Lhuer 2016).
Ein weiterer Vorteil von RPA ist die Erhöhung der Compliance. Dies lässt darauf schließen, dass es für Unternehmen durch den Einsatz von RPA relativ einfach ist, sich an vorgegebene Regeln im Unternehmen zu halten und so ihre Gesamt-Compliance zu verbessern (Ivančić et al. 2019, S. 282). Ein weiterer Vorteil von RPA besteht darin, dass generell keine fortgeschrittenen Programmierkenntnisse benötigt werden, um einfache RPA-Roboter zu erstellen. Hinzu kommt, dass RPA von den Unternehmen ohne größere Anpassungen in die aktuelle IT-Landschaft integriert werden kann, da RPA nur auf dieser aufsetzt (Langmann und Turi 2021, S. 1, 8, 11).
Trotz der aufgezeigten Vorteile von RPA in Bezug auf Effizienzsteigerung und Kostenoptimierung sind auch spezifische Herausforderungen vorhanden. So eignet sich RPA nicht für alle Prozesse. RPA-Roboter sind nicht in der Lage, eigenständig Entscheidungen zu treffen. Aus diesem Grund stellen komplexe, unstrukturierte und stark variierende Prozesse eine Herausforderung für RPA-Roboter dar. Daraus kann abgeleitet werden, dass eine sorgfältige Analyse, Auswahl und Dokumentation der zu automatisierenden Prozesse innerhalb eines Unternehmens zu Beginn essenziell ist (Choi et al. 2021, S. 2 f.; Wanner et al. 2019, S. 2, 5; Langmann und Turi 2021, S. 14-16; Brettschneider 2020, S. 1103). Des Weiteren kann die Instandhaltung der RPA-Roboter für Unternehmen eine signifikante Herausforderung darstellen. RPA-Prozesse erfordern eine kontinuierliche Wartung, da selbst geringfügige Änderungen in den mit dem Prozess verbundenen Systemen und insbesondere in den Oberflächen, oft Anpassungen an den RPA-Robotern notwendig machen. Prozesse, die häufige Anpassungen benötigen, sind besonders fehleranfällig, was zu erhöhtem Wartungs- und Kostenaufwand führen kann (Langmann und Turi 2021, S. 14; Santos et al. 2020, S. 413). Je mehr Roboter im Einsat sind, desto höher wird auch der gesamte Wartungsaufwand innerhalb der RPA-Umgebung für ein Unternehmen (Brettschneider 2020, S. 1107). Durch die Automatisierung von Aufgaben, diezuvorvon menschlichen Mitarbeitern ausgeführt wurden, entfällt zukünftig ein Teil dieser Tätigkeiten aufgrund des Einsatzes von RPA. Daraus resultiert eine Herausforderung im Sinne der Workforce Resilience, d.h. des Widerstands der Mitarbeiter gegenüber dieser Technologie. Es besteht die Gefahr, dass diese befürchten, durch RPA ersetzt zu werden, was zu erhöhten Ängsten hinsichtlich Entlassungen führt. Unternehmen müssen rechtzeitig die Mitarbeiter in den Prozess der RPA-Einführung einbeziehen und ein geeignetes Change Management betreiben. Beispielsweise sollten geeignete Lösungswege und Fortbildungsmöglichkeiten aufgezeigt werden, um die RPA Einführung nicht zu gefährden (Syed et al. 2020, S. 8; Brettschneider 2020, S. 1104 f.; Santos et al. 2020, S. 414; Köhler-Schute 2020, S. 22). Weiterhin lässt sich das Management der Skalierbarkeit als potenzielles Problem identifizieren. So kann es innerhalb von RPA zu Schwierigkeiten kommen, wenn man versucht, eine einzelne Anwendung unternehmensweit auszurollen. Daher sollte bereits zu Beginn der Einführung ein Konzept bezüglich der Infrastruktur erarbeitet werden, um am Ende die gewünschte Skalierbarkeit und Stabilität der RPA-Umgebung erreichen zu können (Syed et al. 2020, S. 12; Langmann und Turi 2021, S. 61).

MICROSERVICES

Definition und Grundlagen

Viele Unternehmen stehen vor der Herausforderung, dass ihre IT-Landschaften nicht mehr zeitgemäß sind. Dies liegt daran, dass sie zu einer Zeit implementiert wurden, als Aspekte wie Modularisierung noch nicht im Fokus standen. Dies erfordert eine Veränderungsbereitschaft, um wettbewerbsfähig bleiben zu können (Dowalil 2018, S. 17 f.; Habibullah et al. 2019, S. 1 f.). Dieser Wandel spiegelt sich insbesondere in der Entwicklung der Software-Architektur wider. In der Vergangenheit waren monolithische Architekturen, bei denen die gesamte Software als ein einziger großer Codeblock umgesetzt wurde, weitverbreitet. Dieser umfangreiche Block erfüllte funktionale und nicht-funktionale Anforderungen, was als Resultat eine enge Verknüpfung zwischen den einzelnen Bestandteilen mit sich brachte. Diese Herangehensweise wird jedoch, besonders bei zunehmender Komplexität, unwirtschaftlich, kostenintensiv und zeitaufwändig. Der Grund dafür liegt in erhöhten Ressourcenaufwänden und verlängerten Entwicklungszeiten. Entsprechend wurden modulare Architekturen entwickelt, um größere Flexibilität und Skalierbarkeit zu erzielen (Yousif 2016, S. 4; Kalske et al. 2018, S. 32 f.). In diesem Zusammenhang spielte die "Service-Oriented Architecture" (SOA) eine bedeutende Rolle. SOA ist eine Architekturform, bei der die Systemlandschaft in einzelne, unabhängige Services zerlegt wird.

Jeder Service ist dabei genau für eine Geschäftsaufgabe zuständig. Diese Services werden innerhalb eines Netzwerks verteilt und kommunizieren über definierte Schnittstellen, wodurch eine lose Kopplung zwischen ihnen entsteht. Unter einer losen Kopplung wird die Minimierung der Verbindungen zwischen einzelnen Softwarekomponenten verstanden. Dies impliziert, dass zwischen den verschiedenen Bestandteilen einer Software möglichst wenige Abhängigkeiten bestehen sollten, damit z. B. in Fehlerfällen nicht das ganze System betroffen ist. Aufbauend auf den Konzepten von SOA haben sich Microservices als eine modulare Unterart bzw. als ein spezifischer Typ einer "Service-Oriented Architecture" entwickelt. (Dowalil 2018, S. 25-29, 121 f., 196; Newman 2020, S. 1). Heutzutage gewinnen Microservices zunehmend an Popularität und werden bereitsvon einer Vielzahl von Unternehmen implementiert (Taibi et al. 2017, S. 23). Wird die Definition eines Microservices betrachtet, so ist festzustellen, dass bisher keine einheitliche Definition existiert. Auf Basis der Definitionen von Fowler und Lewis sowie der Definition von Newman lässt sich jedoch eine allgemeine Charakterisierung ableiten: Microservices sind kleine, unabhängige und lose gekoppelte Einheiten, die jeweils eine spezifische Funktion oder Aufgabe erfüllen. Sie kommunizieren über geeignete Schnittstellen miteinander und sind stark gekapselt, was ihre unabhängige Veröffentlichung und Wartung ermöglicht. Wenn in einem System mehrere solcher Microservices zusammenarbeiten, spricht man von einer Microservice-Architektur. Betrachtet man die Größe eines Microservices näher, so könnte der Begriff „Micro“ eine geringe Größe suggerieren. Allerdings gibt es keine klar definierte Größe für einen Microservice. Die tatsächliche Größe hängt stark vom jeweiligen Kontext ab, weshalb ein einzelner Service nicht zwingend klein sein muss (Fowler und Lewis 2014; Newman 2020, S. 1, 9 f.; Nadareishvili et al. 2016, S. 65 f.) Abbildung 3 zeigt den beispielhaften Aufbau und die Funktionsweise einer solchen.

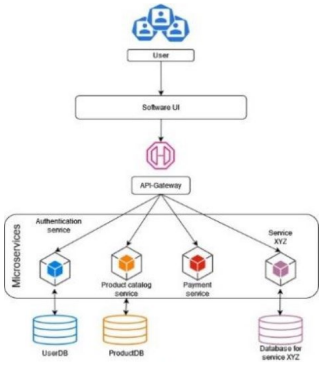

Abbildung 3 Microservice-Architektur Quelle: Eigene Darstellung in Anlehnung an Microsoft Learn o. D., 2023; Lal Sahni 2023

In einer Microservice-Architektur greifen Benutzer über eine Benutzeroberfläche auf die Software zu, die im Hintergrund über ein API-Gateway mit den benötigten Microservices kommuniziert. Das API-Gateway leitet Anfragen weiter und koordiniert die Antworten. Ein Beispiel hierfür ist der Authentifizierungsservice, der den Benutzerlogin als eigenständigen Microservice abwickelt. Microservices sind unabhängig voneinander entwickelbar und skalierbar. Skalierbarkeit erfolgt entweder vertikal durch die Erweiterung der Ressourcen einer Maschine oder horizontal durch die Verteilung der Arbeitslast auf mehrere Systeme (RedHat 2019; AWS o. D.; Blinowski et al. 2022, S. 20360).

Architekturprinzipien

Für die effektive Integration von Microservices in ein System sind bestimmte Architekturprinzipien unerlässlich, um ihr volles Potenzial zu entfalten. Laut der IEEE/ISO/IEC 42010-2022 Norm beschreibt eine Architektur das zentrale Konzept und die charakteristischen Merkmale einer Einheit in einem definierten Umfeld. Sie enthält alle notwendigen Informationen für Implementierung und Weiterentwicklung. Gartner ergänzt diese Definition, indem die verwendete Hardware, Software und Kommunikationsmechanismen als entscheidende Bestandteile der Architektur hervorgehoben werden (IEEE Computer Society/Software & Systems Engineering Standards 2022; Gartner o. D.a).

Das erste zentrale Architekturprinzip von Microservices ist die Modularität. Dabei wird ein System in klar abgegrenzte Module aufgeteilt, die durch ihre externen Eigenschaften, insbesondere ihre Schnittstellen, definiert sind. Diese Schnittstellen ermöglichen die Interaktion zwischen den Modulen. Ein wichtiger Aspekt der Modularität ist die Austauschbarkeit von Modulen. Ein Modul kann durch ein anderes ersetzt werden, solange es dieselben Eigenschaften aufweist. Jedes Modul sollte zudem leicht weiterentwickelbar sein und über eine eigene Dokumentation verfügen (Dowalil 2018, S. 2 f.). Ein weiteres wichtiges Architekturprinzip ist die Kontextgrenze, auch „Bounded Context“ genannt, die aus dem Domain-Driven Design stammt. Dieser Softwareentwicklungsansatz zielt darauf ab, Software entlang der Prozesse und Regeln einer bestimmten Domäne zu entwickeln. Die Kontextgrenze definiert klare Grenzen für ein Modul oder eine Microservice-Komponente und stellt sicher, dass das Modell innerhalb dieser Grenzen einheitlich und konsistent bleibt. Dabei geht es nicht nur um die Bereitstellung einer spezifischen Funktionalität, sondern auch darum, die interne Komplexität zu verbergen, sodass externe Systeme keinen Zugriff auf interne Details erhalten (Newman 2021, S. 52 f., 58 f., 2020, S. 31; Dowalil 2018, S. 64 f.; Fowler 2020). Um die Modularität in einer Architektur effektiv umzusetzen, spielen ebenfalls Software-Design-Prinzipien wie Separation of Concerns und das Single Responsibility Principle eine zentrale Rolle. Das Prinzip der Separation of Concerns besagt, dass jede Funktion eines Systems in einem eigenständigen Baustein realisiert werden sollte. Im Kontext von Microservices bedeutet dies, dass jede Funktionalität als eigenständiger Service abgebildet wird. Das Single Responsibility Principle ergänzt dieses Konzept, indem es sicherstellt, dass jeder Microservice nur eine spezifische Verantwortlichkeit hat. Dadurch existiert für jeden Service nur einen Grund für Änderungen, was die Wartbarkeit und Weiterentwicklung vereinfacht (Dowalil 2018, S. 31

ff.; Hu 2023, S. 107). Das zweite wichtige Architekturprinzip ist die lose Kopplung. Grundsätzlich beschreibt die Kopplung die Abhängigkeiten zwischen den Bausteinen einer Architektur. Im Kontext von Microservices bedeutet lose Kopplung, dass die einzelnen Services nur minimale Informationen übereinander besitzen und weitestgehend unabhängig agieren. Konkret heißt das, dass Änderungen an einem Service keine oder nur minimale Auswirkungen auf andere Services haben sollten. Eine stabile Struktur zeichnet sich dadurch aus, dass sie starke Kohäsion und schwache Kopplung aufweist. Kohäsion beschreibt das Maß, in dem die Komponenten innerhalb eines Moduls eng miteinander verbunden und auf eine gemeinsame Funktionalität ausgerichtet sind. Eine hohe Kohäsion fördert die Wartbarkeit und Weiterentwicklung der Module, da die Funktionen eines Moduls klar definiert und aufeinander abgestimmt sind (Newman 2020, S. 17; Dowalil 2018, S. 25; Newman 2021, S. 38 f.; Farley o. D.). Neben der Modularität und der losen Kopplung ist die Autonomie der Services ein weiteres entscheidendes Prinzip. Die Autonomie lässt sich aus zwei Perspektiven betrachten. Zum einen sollten verschiedene Teams in der Lage sein, im Rahmen einer gemeinsamen Governance (Shared Governance) eigenständig Services zu entwickeln und bereitzustellen. Das bedeutet, dass diese Teams volle Verantwortung für die Entwicklung und Verwaltung ihrer Services tragen. Zum anderen sollten die Microservices selbst ebenfalls autonom sein. Dies erfordert, dass jeder Service so gestaltet ist, dass er unabhängig und in sich geschlossen funktioniert. Dadurch kann ein Microservice losgelöst von anderen Services entwickelt, veröffentlicht und betrieben werden, was die Flexibilität und Skalierbarkeit der gesamten Architektur erheblich steigert (Khan et al. 2021, S. 7). Ein weiteres zentrales Architekturprinzip von Microservices betrifft die Kommunikation zwischen den einzelnen Services. Microservices interagieren über Schnittstellen (APIs), die eine entscheidende Rolle im System spielen, da sie die interne Kommunikation zwischen den Services ermöglichen. Um eine lose Kopplung zu gewährleisten, sollten diese Schnittstellen lediglich die Informationen bereitstellen, die für die Interaktion mit anderen Services notwendig sind. Dies reduziert Abhängigkeiten und fördert die Unabhängigkeit der einzelnen Microservices (Dowalil 2018, S. 123). Im Kontext von Microservices hat sich hier das Designprinzip „smart endpoints and dumb pipes“ durchgesetzt. Jeder Microservice fungiert dabei als eine Art Filter. Er empfängt Anfragen, verarbeitet sie mit der entsprechenden Logik und liefert das entsprechende Ergebnis zurück. Die Kommunikation zwischen den Services wird dabei möglichst einfach gehalten, ohne komplexes Anfragenrouting oder Datentransformationen. Zur Kommunikation werden hauptsächlich zwei Protokolle verwendet: HTTP für synchrone Kommunikation, bei der eine Anfrage und eine direkte Antwort erfolgen, und Lightweight Messaging für asynchrone Kommunikation über einen Nachrichtenbus (Alpers et al. 2015, S. 73; Fowler und Lewis 2014; Dowalil 2018, S. 74 f.; Montemagno et al. 2022). Obwohl das Designprinzip „smart endpoints and dumb pipes“ die Komplexität reduziert und eine einfache Kommunikation zwischen den Services fördert, bleibt die Möglichkeit von Fehlern bestehen. Diese können durch menschliches Versagen oder technische Störungen verursacht werden. Daher ist Resilienz ein zentraler Aspekt beim Entwurf einer Microservice-Architektur. Resilienz beschreibt die Fähigkeit eines Systems, trotz auftretender Fehler weiter zu funktionieren und sich schnell zu erholen. Das Ziel ist es, dass bei einem Fehler nicht das gesamte System ausfällt, sondern nur die betroffenen Bereiche beeinträchtigt werden, während der Rest des Systems weiterhin funktionsfähig bleibt. Zur Unterstützung der Resilienz können Mechanismen wie Timeouts implementiert werden. Dabei wird festgelegt, dass eine Serviceanfrage innerhalb einer bestimmten Zeit beantwortet werden muss. Bleibt diese aus, wird der Service als ausgefallen betrachtet (Indrasiri und Siriwardena 2018, S. 42; Wolff 2018, S. 207 f.). Damit ein reibungsloser Betrieb in einer Microservice-Architektur gewährleistet werden kann, sind Monitoring- und Logging-Funktionalitäten unerlässlich. Logging ermöglicht es, Ereignisse in den einzelnen Services nachzuvollziehen, was nicht nur für die Erstellung von Statistiken, sondern vor allem für die effiziente Fehlersuche von zentraler Bedeutung ist. Log-Dateien sind in der Regel so strukturiert, dass sie von Menschen leicht interpretiert werden können. Durch Monitoring können wichtige Metriken wie die Antwortzeiten der Services und die Anzahl fehlgeschlagener Anfragen kontinuierlich überwacht werden. Diese Informationen sind entscheidend, um frühzeitig Probleme zu identifizieren und die Systemstabilität sicherzustellen (Wolff 2018, S. 244 f.; Indrasiri und Siriwardena 2018, S. 48 f., 373; Khan et al. 2021, S. 8). Aufgrund der erhöhten Netzwerkkommunikation in einer Microservice-Architektur ist es essenziell, dass nur autorisierte Parteien Zugriff auf die Services haben. Daher sollten Microservices nur die minimal benötigten Rechte besitzen, insbesondere bei Datenbankzugriffen und sensiblen Ressourcen. Ein effizientes Identity- und Access Management (IAM) ist unerlässlich, um sicherzustellen, dass ausschließlich autorisierte Nutzer und Services auf die Microservices zugreifen können. Zudem müssen Zugangsdaten wie E-Mails und Passwörter sicher gespeichert werden und dürfen nicht im Klartext vorliegen, um Sicherheitsrisiken zu vermeiden (Newman 2021, S. 29, 345-347, 354-456).

Vorteile und Herausforderungen

Microservices bieten gegenüber monolithischen Architekturen zahlreiche Vorteile, jedoch besitzen sie auch Herausforderungen. Ein wesentlicher Vorteil besteht in der Unterstützung agiler Arbeitsweisen, die es Unternehmen ermöglicht, schneller auf veränderte Geschäftsanforderungen zu reagieren. Innerhalb einer Microservice-Architektur lassen sich neue Services leichter entwickeln, bereitstellen und testen, was eine höhere Flexibilität sowie schnellere Iterationen ermöglicht. Falls ein neu entwickelter Service nicht den gewünschten Anforderungen entspricht, kann dieser mit geringem Aufwand deaktiviert und durch eine alternative Implementierung ersetzt werden. Diese Agilität verkürzt die Entwick-

lungszeit von Microservices erheblich, was es den Entwicklerteams ermöglicht, flexibler und schneller auf Marktveränderungen zu reagieren. Dadurch können Unternehmen ihre Innovationszyklen beschleunigen und Wettbewerbsvorteile erzielen (Indrasiri und Siriwardena 2018, S. 14; Nadareishvili et al. 2016, S. 14-16; Khan et al. 2021, S. 11). Ein weiterer Vorteil von Microservices ist die verbesserte Wartbarkeit. Dank der modularen und unabhängigen Struktur lassen sich Services leichter warten. Da die Logik und Datenhaltung innerhalb der Services erfolgt, sind keine externen Abhängigkeiten notwendig. Updates betreffen somit nur den jeweiligen Service, während der Rest des Systems unverändert bleibt. Dies fördert nicht nur die Wartbarkeit, sondern auch die langfristige Nachhaltigkeit der Softwareentwicklung, da veraltete Systemstrukturen vermieden und eine zukunftsfähige IT-Infrastruktur geschaffen wird (Khan et al. 2021, S. 11; Eyerman und Hur 2022, S. 1; Wolff 2018, S. 60-62). Neben der Wartbarkeit ermöglichen Microservices Unternehmen, neue Features und Bugfixes schneller zu veröffentlichen. Die Unabhängigkeit der Entwicklungsteams trägt dazu bei, da sie ohne Abhängigkeit von anderen Teams effizienter arbeiten können. Zudem erlaubt die Microservice-Architektur den Entwicklern, moderne Technologien zu nutzen, ohne an veraltete Entscheidungen gebunden zu sein. Ein weiterer Vorteil ist die Skalierbarkeit: Microservices können individuell skaliert werden, ohne das gesamte System anpassen zu müssen. Dies ermöglicht eine effiziente Nutzung von Ressourcen, indem Services bei steigenden Anfragen hoch- und bei sinkenden Anfragen wieder herunterskaliert werden, ohne die Performance zu beeinträchtigen (Khan et al. 2021, S. 11, 12; Wolff 2018, S. 64 f., 66 f.; GitLab 2022). Trotz der Vorteile wie gesteigerte Agilität, verbesserte Wartbarkeit, verkürzte Time-to- Market und erhöhter Skalierbarkeit müssen auch die Herausforderungen von Microservices berücksichtigt werden. Eine zentrale Herausforderung ist die erhöhte Komplexität, die durch die Vielzahl autonomer und lose gekoppelter Komponenten entsteht. Insbesondere die Inter-Service-Kommunikation erfordert effiziente und zuverlässige Kommunikationswege, die oft komplexer sind als die Entwicklung der Services selbst. Auch das Daten- und Transaktionsmanagement wird durch die verteilte Logik und Datenhaltung anspruchsvoller. Zusätzlich ist bereits die Definition und Erstellung der Services eine Herausforderung. Um eine effektive Modularisierung sicherzustellen, müssen Unternehmen klar identifizieren, welche Funktionalitäten als eigenständige Module umgesetzt werden sollten. Falsch definierte Servicegrenzen können zu erhöhtem Datenaustausch über das Netzwerk führen, was wiederum die Kopplung zwischen den Services verstärkt und der Grundidee der losen Kopplung widerspricht (Indrasiri und Siriwardena 2018, S. 15 f.; Jamshidi et al. 2018, S. 31).

Auch das Monitoring der Services stellt eine Herausforderung dar. Durch die Verteilung der Microservices kann es schwierig sein, den Überblick über die Beziehungen zwischen den einzelnen Services zu behalten. Die autonome Struktur der Services, die zudem in unterschiedlichen Technologien implementiert sein können, macht ein umfassendes Logging- und Monitoringkonzept unerlässlich. Dies wird zusätzlich dadurch erschwert, dass Microservices oft von unterschiedlichen Teams entwickelt werden. Eine weitere Herausforderung sind die Kosten bei der Einführung einer Microservice-Architektur, die vor allem in der Anfangsphase hoch ausfallen können. Diese resultieren unter anderem aus der Notwendigkeit, die bestehende Infrastruktur zu erweitern, etwa durch den Ausbau des Netzwerks, zusätzlichen Speicherplatz oder die Implementierung zusätzlicher Software. Zudem müssen während der Migration die monolithischen Systeme parallel zu den neuen Microservices betrieben werden, was den Aufwand weiter erhöht. Ein weiterer Faktor sind die verzögerten Entwicklungsprozesse zu Beginn der Umstellung, da sich Entwickler zunächst an die neuen Strukturen und Arbeitsabläufe gewöhnen müssen. Abhängig vom Erfahrungsgrad der Entwickler können zusätzliche Kosten für die Einstellung von Fachkräften mit Microservice-Kenntnissen entstehen. Trotz dieser Anfangskosten sollten Microservices jedoch als strategische Investition betrachtet werden, die langfristig Effizienzsteigerungen und Flexibilität fördern (Niedermaier et al. 2019, S. 42-44; Khan et al. 2021, S. 14; Baškarada et al. 2020, S. 6; Newman 2021, S. 26-28; Singleton 2016, S. 17).

USECASE

Vorgehensweise Experteninterviews

Zur Erfassung der Anforderungen an die zu entwickelnde Microservice-Architektur wurden leitfadenbasierte Experteninterviews durchgeführt. Ziel dieser Interviews war es, relevante Anforderungen zu identifizieren und wertvolle Einblicke aus der Praxis zu gewinnen, die aus anderen Quellen nur schwer zu ermitteln wären. Der Interviewleitfaden ist in drei Bereiche unterteilt:

1. Allgemeine Informationen der Experten: Zunächst wurden Fragen gestellt, um den Hintergrund der Experten zu erfassen und ihre Erfahrungen im jeweiligen Unternehmenskontext besser einzuordnen.
2. Kenntnisse über Microservices und Robotic Process Automation (RPA): In diesem Abschnitt wurden die Experten zu ihrer Einschätzung hinsichtlich verschiedener relevanter Themenbereiche innerhalb einer Microservice-Architektur befragt. Dabei sollten sie die Ziele, die mit einer solchen Architektur verfolgt werden können, nach Wertbeitrag und Realisierbarkeit priorisieren, um die Anforderungen praxisnah und umsetzbar zu gestalten.
3. Spezifische Anforderungen an die Architektur: Abschließend wurde ein tieferer Einblick in die spezifischen Erwartungen und Anforderungen der Experten gewonnen. Hierbei lag der Fokus auf den wichtigsten Mehrwerten für die zukünftige Architektur sowie auf potenziellen Methoden zur Realisierung und Katalogisierung der Microservices.

Die Interviews wurden per Videokonferenz durchgeführt, um eine strukturierte Erhebung der Anforderungen

sicherzustellen.

Ergebnisse der Experteninterviews

Die befragten Experten setzten sich aus Beratern und Entwicklern zusammen, was zu unterschiedlichen Ansichten führte. Die Interviews zeigten, dass das Verständnis von Microservices und RPA variiert. Einige Experten sehen RPA als Werkzeug zur Automatisierung von Frontend-Prozessen, während andere eine breitere Definition verwenden, die auch die Automatisierung von Prozessen in Systemen ohne native Automatisierungsfunktionen umfasst. Beim Thema Microservices reichten die Auffassungen von einer einfachen modularen Architektur bis hin zur Weiterentwicklung der serviceorientierten Architektur (SOA), die komplexere Systemarchitekturen ermöglicht. Ein zentrales Ergebnis der Interviews ist die Identifikation der Hauptziele bei der Implementierung einer Microservice-Architektur im RPA-Umfeld. Dabei wurden Skalierbarkeit, Flexibilität, Wartbarkeit und Performance als zentrale Prioritäten genannt. Außerdem bewerteten die Experten die potenziellen Mehrwerte nach Wertbeitrag und Realisierbarkeit, was eine systematische Rangordnung ermöglichte. Besonders hoch evaluiert wurden die Mehrwerte „Wiederverwendbarkeit“, „Wartbarkeit“, „Skalierbarkeit“, „Ersetzbarkeit“ und „Flexibilität“. Diese spielen eine entscheidende Rolle bei der Architektur entwicklung (siehe Tabelle 1). Ein weiterer Schwerpunkt der Untersuchung lag auf der Analyse, wie die einzelnen Expertengruppen, insbesondere Entwickler und Berater, die Mehrwerte unterschiedlich bewerten. Hier zeigte sich eine weitgehende Übereinstimmung der Top-5-Mehrwerte beider Gruppen. Jedoch wurden Unterschiede hinsichtlich der Gewichtung einzelner Mehrwerte beobachtet: Während Entwickler vermehrt die „Stabilität“ der Systeme in den Vordergrund stellten, legten Berater einen stärkeren Fokus auf die „Flexibilität“ der Architektur.

Tabelle 1 Top-Mehrwerte Quelle: Eigene Darstellung

Reihenfolge	Top-Mehrwerte	Gesamtpunktzahl durch Experten
1	Wiederverwendbarkeit	675
2	Wartbarkeit	544
3	Skalierbarkeit	501
4	Ersetzbarkeit	484
5	Flexibilität	480
6	Testbarkeit	428
7	Performance	403
8	Stabilität	396
9	Fehlerbehebungsmöglichkeiten	351
10	Integrationsfähigkeit	224
11	Time-to-Market	146
12	Compliance und Governance	128
13	Administrierbarkeit	104
14	Sicherheit	92

Die detaillierte Analyse der Mehrwertbewertungen verdeutlicht, dass insbesondere die „Wiederverwendbarkeit“ und „Wartbarkeit“ von allen Experten als sehr relevant eingestuft wurden. Dies lässt sich darauf zurückführen, dass diese Eigenschaften eine langfristige Kostenersparnis und höhere Effizienz bei der Wartung und Erweiterung der Architektur ermöglichen. Die Unterschiede zwischen Entwicklern und Beratern in der Priorisierung anderer Merkmale, wie „Stabilität“ versus „Flexibilität“, lassen sich durch die jeweiligen Rollen und Verantwortlichkeiten der Experten erklären. Entwickler fokussieren sich in ihrer Arbeit häufig auf die technische Umsetzbarkeit und die Gewährleistung der Systemstabilität, während Berater strategische Faktoren wie Anpassungsfähigkeit und Skalierbarkeit im Kontext zukünftiger Anforderungen stärker gewichten.

Zusammenfassend lässt sich festhalten, dass die Experteninterviews wertvolle Einblicke in die unterschiedlichen Einschätzungen und Prioritäten hinsichtlich der Mehrwerte von Microservices und RPA liefern. Die Ergebnisse deuten darauf hin, dass trotz unterschiedlicher beruflicher Hintergründe zentrale Mehrwerte von allen Expertengruppen ähnlich hoch gewichtet werden. Dies legt nahe, dass bestimmte Eigenschaften, wie Wartbarkeit und Wiederverwendbarkeit, als universell bedeutend für die Implementierung von Microservice-Architekturen in Verbindung mit RPA gelten.

Anforderungen und Anpassung der Architekturprinzipien

Die abgeleiteten Anforderungen an eine Microservice-Architektur im RPA-Umfeld werden auf Basis eines kategorienbasierten Ansatzes abgeleitet. In der Kategorie „Entwicklung und Design“ stehen Modularität, Wiederverwendbarkeit, Flexibilität und lose Kopplung im Vordergrund, um die Anpassungsfähigkeit und Effizienz der Architektur zu maximieren. Im Bereich „Betrieb und Wartung“ werden die Wartbarkeit, einfache Verwaltung und umfassende Dokumentation hervorgehoben, um eine langfristig stabile und pflegeleichte Architektur sicherzustellen. Für die Kategorie „Performance und Qualität“ waren vor allem Skalierbarkeit, Stabilität und Robustheit von zentraler Bedeutung, da diese eine hohe Effizienz und Zuverlässigkeit auch unter variierenden Bedingungen garantierten. In der Kategorie „Governance und Compliance“ liegt der Fokus auf der Entwicklung von Richtlinien, einer Katalogisierung der Microservices sowie der Verantwortung für die Einhaltung von Standards und deren regelmäßiger Überprüfung.

Die zuvor beschriebenen Anforderungen werden durch angepasste Architekturprinzipien in der Microservice-Architektur für RPA abgebildet. Da RPA-Microservices noch einen neuen Ansatz darstellen, müssen die allgemeinen Prinzipien für Microservices an die spezifischen Gegebenheiten angepasst werden.

Die Modularität ist so anzupassen, dass die RPA-Microservices flexibel, leicht austauschbar und wiederverwendbar sind. In der Architektur wird dies dadurch umgesetzt, dass jedes Modul über wenige Parameter (wie Modulname und Inputparameter) aufgerufen werden kann. Durch diese Struktur können einzelne Module nahtlos durch andere mit denselben Eigenschaften ersetzt werden.

Das Prinzip des Bounded Contexts wird übernommen, um sicherzustellen, dass die Komplexität der Automatisierungsprozesse verborgen bleibt. In der Architektur wird dies durch die präzise Abgrenzung der RPA-Microservices erreicht, sodass jeder Service eine spezifische Aufgabe übernimmt, ohne Details seiner internen Funktionsweise nach außen preiszugeben. Dies verstärkt die Anwendung der Prinzipien Separation of Concerns

und Single Responsibility: Diese stellen sicher, dass jeder Microservice eine klar abgegrenzte Funktion erfüllt. In der Architektur bedeutet dies, dass RPA-Microservices als isolierte, unabhängig operierende Einheiten entworfen werden, was die Wartbarkeit und Integration erleichtert.
Die lose Kopplung wird dahingehend angepasst, dass RPA-Microservices möglichst unabhängig voneinander agieren. In der Architektur erfolgt die Umsetzung durch die Schaffung von isolierten Automatisierungsprozessen, die nur minimale Abhängigkeiten zu anderen Prozessen aufweisen. Dies ermöglicht es, Änderungen an einem RPA-Prozess vorzunehmen, ohne dass andere Prozesse oder der Gesamtprozessfluss beeinträchtigt werden, solange die definierten Schnittstellen (Input- und Outputparameter) unverändert bleiben. Trotz der engen Integration mit den zu automatisierenden Systemen bleibt die lose Kopplung durch die klare Trennung der Automatisierungsaufgaben bestehen.
Die Autonomie der Services wird ebenfalls angepasst. Jeder RPA-Service ist so konzipiert, dass er unabhängig von anderen Services betrieben und veröffentlicht werden kann. In der Architektur wird dies durch den Einsatz von einer Orchestrierungssoftware unterstützt, die die Koordination der RPA-Services ermöglicht. Dadurch können die RPA-Services flexibel in unterschiedliche Geschäftsprozesse integriert und wiederverwendet werden, was die Autonomie und Skalierbarkeit der Architektur erhöht. Die Prinzipien der Resilienz, Monitoring und Logging werden so angepasst, dass sie den spezifischen Anforderungen von RPA entsprechen. In der Architektur erfolgt die Abbildung durch die Nutzung von RPA-Plattformen und Orchestrierungsumgebungen, die Mechanismen zur Überwachung des Zustands und der Performance der RPA-Prozesse bieten. Diese Systeme ermöglichen es, Fehlerszenarien aufzuzeichnen und die Stabilität der Microservices sicherzustellen. Logging und Monitoring werden genutzt, um Aktivitäten nachzuvollziehen und für Compliance- und
Audit-Zwecke zu protokollieren. Beim Identity Management wird das Prinzip angepasst, um eine sichere Verwaltung von Zugriffsrechten in der Architektur zu gewährleisten. Die Umsetzung erfolgt durch die in RPA- und Orchestrierungsplattformen integrierten Lösungen, die sicherstellen, dass nur autorisierte Nutzer und Systeme auf bestimmte Ressourcen zugreifen können. Die sichere Speicherung von Zugangsdaten erfolgt dabei durch verschlüsselte Verfahren, um die Sicherheit der Architektur weiter zu erhöhen.

Microservice-Architekturentwurf

Aufbau und Komponenten

Die entwickelte Microservice-Architektur bildet die Grundlage für die Modularisierung von RPA-Prozessen und zielt darauf ab, eine flexible, skalierbare, modulare, ersetzbare, wiederverwendbare und wartbare Umgebung zu schaffen.
Die Architektur unterscheidet zwischen zwei Arten von Microservices:

- Wertgenerierende Microservices: Diese Services verarbeiten Daten nach festen Regeln und liefern ein messbares Ergebnis, z. B. das Verarbeiten einer Excel-Tabelle und das Bereitstellen eines neuen Outputs.
- Nicht-wertgenerierende Microservices: Diese Services unterstützen den Gesamtprozess, tragen jedoch nicht direkt zum Endergebnis bei, wie z. B. die Authentifizierung an Systemen.

Im RPA-Kontext ist diese Unterscheidung wichtig, da nach jedem abgeschlossenen Prozess eine erneute Authentifizierung erforderlich ist. Die Architektur sieht daher vor, nicht-wertgenerierende Services als standardisierte Module bereitzustellen, die zentralisiert angepasst werden können.
Wertgenerierende Microservices werden hingegen als eigenständige RPA-Prozesse umgesetzt. Jeder Prozess fungiert als eigenständiger Microservice, der spezifischen Input verarbeitet und daraufhin einen entsprechenden Output generiert. Diese Services werden über ein Orchestrierungstool genutzt, wobei die Gesamtprozessmodellierung in einem Business Process Automation Tool erfolgt. Die Kommunikation erfolgt dabei über zuvor vordefinierte Schnittstellen, die den Prozessoutput zurückgeben.

Katalogisierung

Die Anzahl der Microservices kann in Unternehmen schnell steigen, besonders bei dezentralen Organisationsstrukturen. Ein zentraler Katalog bietet hierbei eine hilfreiche Übersicht zu Funktionalitäten und Zuständigkeiten der Microservices. Der ausgearbeitete Katalog dient der Verwaltung und Weiterentwicklung und basiert auf den geführten Experteninterviews. Zu den zentralen Katalogelementen gehören grundlegende Informationen wie der Servicename, eine klare Beschreibung der Entwickler und Verantwortlichen. Diese Daten helfen, Zuständigkeiten zu klären und SLAs festzulegen. Technische Details wie Inputparameter, erwarteter Output und Abhängigkeiten ermöglichen eine klare Nutzungsübersicht. Wichtig ist es, dort die aktuelle Version sowie die letzte Änderung zu dokumentieren, inklusive einer Verlinkung zu weiterführenden Informationen (z.B. einer Wiki-Seite). Für die Umsetzung bietet sich eine Tabellenstruktur in einer Wiki-Umgebung wie Confluence an, um die Übersichtlichkeit zu wahren. Mit steigender Anzahl an Microservices wird die Nutzung eines Marktplatzes empfohlen. Ein Beispiel ist der UiPath-Marktplatz, bei dem Services nach Kategorien filterbar sind. Eine mögliche Umsetzung kann z.B. mit dem Open-Source-Tool „Backstage“ von Spotify erfolgen, das über einen zentralen Softwarekatalog Metadaten und Abhängigkeiten von Microservices verwaltet und visualisieren kann.

Governance und Maintenance

Governance definiert die Richtlinien und Standards, die für den Betrieb und die Verwaltung der Architektur notwendig sind. Wartung bezieht sich auf die kontinuierliche Aktualisierung und Optimierung der Microservices, um

deren Leistung und Zuverlässigkeit sicherzustellen. Es wird eine Shared Governance empfohlen, bei der Teams innerhalb eines festgelegten Rahmens selbstständig Entscheidungen treffen können und Verantwortung für ihre Microservices übernehmen. Dabei sind klare Standards, insbesondere in Bezug auf Sicherheit, Dokumentation und Zugriffsrichtlinien, von zentraler Bedeutung. Jeder Microservice sollte modular, skalierbar und nur mit dem notwendigen Funktionsumfang ausgestattet sein. Bereits bestehende Monitoring- und Logging-Systeme sollten genutzt und nach einer Testphase bewertet werden. Um die Einhaltung der Standards sicherzustellen, wird die Einrichtung eines Governance-Komitees vorgeschlagen, das als zentrale Anlaufstelle fungiert und die Implementierung der Richtlinien überwacht. Microservices sind regelmäßig zu überprüfen und zu optimieren, beispielsweise je nach Art des Services halbjährlich oder jährlich. Diese Überprüfungen sollten durch automatisierte Tests unterstützt werden, die regelmäßig erfolgen. Zusätzlich wird empfohlen, Schulungs- und Weiterbildungsprogramme für Mitarbeiter anzubieten, um ein tieferes Verständnis von Microservices, den zugrundeliegenden Technologien und Best Practices im gesamten Unternehmen zu fördern. Dies trägt nicht nur zur Verbesserung der fachlichen Kompetenz bei, sondern unterstützt auch die fortlaufende Optimierung der Architektur.

Integration

Die bestehende Beispiel-Infrastruktur nutzt UiPath als RPA-Tool und Flowable als BPA-Tool. Diese beiden Systeme spielen eine zentrale Rolle bei der Integration im Bebauungsplan.

Die Architektur unterscheidet zwischen wertgenerierenden und nicht-wertgenerierenden Microservices. Für wertgenerierende Microservices wird ein vierstufiges Modell verwendet, das aus den Schichten Workflow, Orchestrator, Automation Control Center und Automation Bot besteht. Die Workflow-Schicht stellt die Gesamtprozessmodellierung in Flowable dar, wo Prozesse nach dem BPMN 2.0-Standard modelliert werden. UiPath Microservices werden in den Workflow integriert und lassen sich über eine Service-Task ansteuern. Diese Service-Task kommuniziert über einen Konnektor mit dem UiPath Orchestrator und führt API-Aufrufe aus, um den entsprechenden RPA-Prozess zu starten. Der Konnektor informiert bei Fehlern und bietet eine Restart-Option, um die Robustheit der Prozesse zu gewährleisten. Sobald der UiPath Orchestrator den Aufruf erhält, wird ein Warteschlangenobjekt angelegt. Die Automation Control Center Schicht verarbeitet Prozesse in einer Warteschlange, um die begrenzte Anzahl an Robotern optimal nutzen zu können. Dadurch wird verhindert, dass Prozesse gleichzeitig gestartet werden und zu Systemabstürzen führen. Die Priorisierung von Prozessen ist ein integraler Bestandteil, wobei zehn Prioritätsstufen zur Verfügung stehen, von „kritisch“ bis „niedrig“. Dies ermöglicht eine effiziente und gesteuerte Bearbeitung der Aufgaben. Sobald ein Prozess ausgeführt wird, erfolgt die Rückgabe der Ergebnisse an den Orchestrator, der die Informationen an Flowable weiterleitet. Durch diese Rückmeldung kann der Workflow fortgesetzt werden. Weiterhin sind Parallelisierungen innerhalb des Modells möglich, um die Effizienz der Prozesse zu maximieren. Abbildung 4 zeigt den Aufbau der Microservice-Architektur für wertgenerierende Microservices.

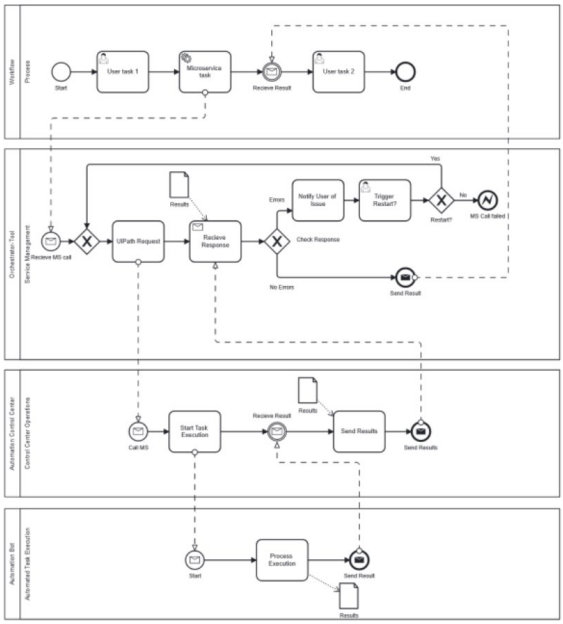

Abbildung 4 Wertgenerierende Microservice-Architektur Quelle: Eigene Darstellung

Der zweite Teil der Microservice-Architektur (Abbildung 5) konzentriert sich auf die Bereitstellung nichtwertgenerierender Microservices in Form von wiederverwendbaren Modulen. Diese Module erfüllen unterstützende Funktionen, die zwar keinen direkten Mehrwert schaffen, aber für die Umsetzung von Geschäftsprozessen notwendig sind wie z.B. System-Authentifizierungen. Eine effiziente Implementierung dieser Module lässt sich durch den Einsatz von UiPath-Bibliotheken realisieren, die eine standardisierte Bereitstellung solcher Prozesse ermöglichen und somit zur Konsistenz und Wiederverwendbarkeit in der gesamten Architektur beitragen. Die Architektur der Bibliotheken besteht aus zwei zentralen Bereichen: der Entwicklung und der Aktualisierung. Zunächst werden die Module als eigenständige Prozesse innerhalb einer Bibliothek entwickelt, wobei jedes Modul eine spezifische Funktionalität abbildet, die es in den Geschäftsprozessen erfüllen soll. Ein Beispiel hierfür ist ein Authentifizierungsprozess, der als eigenständiges Modul in eine Bibliothek integriert wird. Diese Modularisierung ermöglicht eine flexible Wiederverwendung der Prozesse in unterschiedlichen Kontexten. Nach der Entwicklung werden die Module umfassend getestet, um sicherzustellen, dass sie fehlerfrei funktionieren. Anschließend wird die Bibliothek veröffentlicht, sodass die Module in weiteren Prozessen importiert und verwendet werden können. Um eine flexible Handhabung der Module zu gewährleisten, lassen sich Parameter wie Argumente, Assets oder Dropdown-Menüs definieren, durch die notwendige Informationen dynamisch in das Modul übergeben werden. Dies erhöht die Flexibilität bei der Anwendung der Module, da sie ohne Änderungen an ihrer Struktur in

verschiedenen Prozessen genutzt werden können. Der zweite wesentliche Aspekt der Architektur betrifft den Aktualisierungsprozess der Bibliotheken. Im RPA-Umfeld können selbst kleine Änderungen an Systemoberflächen zu Fehlern führen, weshalb eine regelmäßige Anpassung und Wartung der Module erforderlich ist. Um dies effizient zu gestalten, müssen sich Bibliotheken einfach aktualisieren lassen. Dazu wird das Modul in der Bibliothek angepasst, erneut getestet und nach erfolgreicher Prüfung erneut veröffentlicht. Anschließend erfolgt die Aktualisierung aller Prozesse, welche die Bibliothek nutzen, mithilfe eines Massenupdatetools. Dieses Tool ermöglicht es, alle betroffenen Prozesse automatisch auf die neueste Version der Bibliothek zu aktualisieren, wodurch der manuelle Aufwand und die Fehleranfälligkeit erheblich reduziert werden. Insgesamt ermöglicht die Bereitstellung und Verwaltung wiederverwendbarer Module in einer Microservice-Architektur eine hohe Flexibilität und Effizienz. Durch die klare Trennung zwischen wertgenerierenden und unterstützenden Prozessen wird die Architektur modularer und leichter wartbar. Die Verwendung von Bibliotheken spielt hierbei eine zentrale Rolle, da sie nicht nur die Wiederverwendbarkeit und Standardisierung fördern, sondern auch die unkomplizierte Wartung und Aktualisierung von Prozessen in dynamischen Umgebungen sicherstellen.

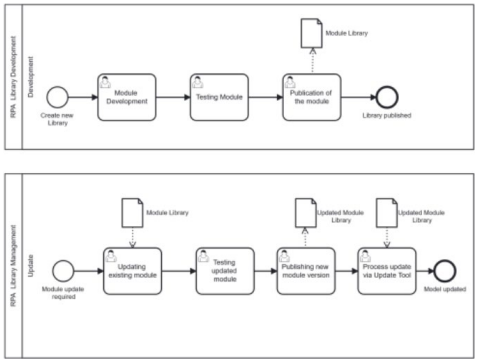

Abbildung 5 Nicht-wertgenerierende Microservice Architektur Quelle: Eigene Darstellung

Bewertung der Use Case Architektur

Für die Evaluierung der Microservice-Architektur wird das visuelle Hilfsmittel der Harvey Balls verwendet. Diese ermöglichen eine differenzierte Darstellung des Erfüllungsgrads der zuvor erhobenen Anforderungen. Ein leerer Kreis (○) steht für Nichterfüllung, ein vollständig ausgefüllter Kreis (●) steht für die vollständige Erfüllung. Zwischen diesen beiden Extremen ermöglichen die Abstufungen von 25 % (◔), 50 % (◑) und 75 % (◕) eine feinere Differenzierung der Erfüllungsgrade an die Architektur.

Tabelle 2 Harvey Balls Skala Quelle: Eigene Darstellung

Skala	Erfüllungsgrad	Beschreibung
○	*Nicht erfüllt (0 %)*	Die Architektur bietet aktuell keine Unterstützung, um diese spezifische Anforderung zu erfüllen.
◔	*Ansatzweise erfüllt (25 %)*	Es existieren initiale Eigenschaften oder Komponenten in der Architektur, durch welche die Anforderung in einem grundlegenden Umfang erfüllt werden, jedoch sind weitere Entwicklungen und Optimierungen notwendig.
◑	*Teilweise erfüllt (50 %)*	Die Architektur erfüllt diese Anforderung in einigen, jedoch nicht allen Aspekten. Die aktuelle Lösung adressiert die grundlegenden Bedürfnisse, doch es besteht noch erheblicher Bedarf an Verbesserungen.
◕	*Überwiegend erfüllt (75 %)*	Die Architektur erfüllt die Anforderung in den meisten Punkten effektiv. Kleinere Optimierungen oder Erweiterungen könnten jedoch noch implementiert werden, um eine vollständige und umfassendere Lösung zu bieten.
●	*Vollständig erfüllt (100 %)*	Die Architektur erfüllt die Anforderung vollständig und zuverlässig mit allen notwendigen Funktionen, die zur optimalen Erfüllung benötigt werden. Keine weiteren Verbesserungen sind erforderlich.

Nachfolgend werden die einzelnen Bewertungen kurz erläutert:

Wiederverwendbarkeit (●): Die Architektur unterstützt konkret zwei Arten von Wiederverwendbarkeit. Zum einen die Wiederverwendbarkeit in Form von Bibliotheken, zum anderen die Wiederverwendbarkeit von ganzen Services. Hierdurch unterstützt die vorgestellte Architektur die Anforderung der Wiederverwendbarkeit vollumfänglich.

Wartbarkeit (◕): Die vorliegende Architektur erleichtert die Wartbarkeit der Prozesse. Zukünftig muss nur noch an einer Stelle die Bibliothek oder der Service geändert werden, anstatt jeden Prozess einzeln anzupassen. Im Kontext von Bibliotheken ist zwar weiterhin die Verwendung eines Update Tools notwendig, dennoch wird die Wartbarkeit durch die Architektur deutlich verbessert.

Skalierbarkeit (◕): Die Skalierbarkeit wird durch die Verwendung von Warteschlangen optimiert, dennoch ist die Skalierbarkeit primär im UiPath Kontext durch die begrenzte Roboterverfügbarkeit bzw. durch das Lizenzmodell weitestgehend eingeschränkt.

Ersetzbarkeit (●): Die Architektur besitzt durch das Konzept der Microservices eine adäquate Ersetzbarkeit. Es ist möglich einzelne Services in der Flowable Prozesskette oder Bibliotheksmodule innerhalb der Ui-Path Prozesskette auszutauschen. Die einzige Voraussetzung für einen anpassungsfreien Austausch ist, dass Input und Output identisch sind.

Flexibilität (◕): Die Architektur zeigt sich anpassungsfähig durch die Unterscheidung zwischen wertgenerierenden und nicht-wertgenerierenden Microservices, sowie durch die Verwendung eines Orchestrierungstools wie Flowable. Diese Strukturierung ermöglicht es, zukünftig auch Microservices anderer Systeme zu integrieren, was die Flexibilität weiter steigert. Es ist allerdings zu beachten, dass starre/feste Prozessabläufe und zu spezifisch gestaltete Prozesse die Flexibilität einschränken können.

Testbarkeit (◑): Die Architektur fördert und erweitert die Testbarkeit, indem sie das separate Testen einzelner Module ermöglicht. Dies bedeutet, dass man jeden Microservice individuell und isoliert auf seine Funktionalität prüfen kann, was zu einer hohen Testbarkeit führt. Jedoch ist unklar, wie Integrationstests also Test, die auf das Zusammenspiel zwischen den einzelnen Microservices

innerhalb des Systems abzielen, realisiert werden können. Dies erweist sich als relevant, da unterschiedliche Systeme innerhalb der Prozesse verwendet werden.

Performance (◑): Die Performance lässt sich nicht genau bewerten, da dies über einen längeren Zeitraum beobachtet und anhand der dadurch gewonnenen Daten bewertet werden sollte. Dennoch stellt die Verwendung von Warteschlangen innerhalb der Architektur eine praktikable Lösung für die Lastenverteilung dar, was auf eine gute Performance hindeuten kann.

Stabilität (◑): Die Stabilität der Architektur ist vorrangig an die Stabilität der Systeme und Prozesse gebunden, weshalb sich eine Bewertung ebenfalls als schwierig erweist. Die Architektur ist offen für Mechanismen, welche die Stabilität fördern, wie der bereits eingebaute Restart Mechanismus im Fehlerfall. Solche Mechanismen sind wichtig, um bei Ausfällen oder Fehlern die Funktionalität der Prozesse aufrecht zu erhalten und schnelle Wiederanläufe zu ermöglichen. Allerdings müssen weitere Mechanismen eingeführt werden, um die Stabilität sicherzustellen.

Fehlerbehebungsmöglichkeiten (◑): Die Fehlerbehebungsmöglichkeiten sind durch den implementierten Restart-Mechanismus innerhalb der Architektur berücksichtigt worden, was die Resilienz gegenüber Störungen erhöht. Diese Möglichkeiten sind jedoch in hohem Maße von den spezifischen Prozessen abhängig. Daraus folgt, dass ohne detaillierte Kenntnisse über die einzelnen Prozessabläufe eine vollständige Bewertung der Fehlerbehebungsfähigkeit nicht möglich ist.

Integrationsfähigkeit (●): Die Architektur bietet eine hohe Integrationsfähigkeit, was durch die klar definierte Schnittstelle zwischen dem Orchestrierungstool und der RPA-Umgebung gewährleistet wird. Zudem ermöglicht die modulare Gestaltung eine flexible Einbindung neuer Dienste und die Erweiterung bestehender Funktionalitäten ohne größere Änderungen an der Gesamtarchitektur. Die Offenheit der Architektur erleichtert die Integration von Drittanbieter-Software und die Anpassung an geänderte Geschäftsanforderungen. Allerdings setzt die vollständige Ausschöpfung der Integrationsfähigkeit eine tiefergehende Prüfung der Kompatibilität und eine fehlerfreie Konfiguration der einzelnen Komponenten und Schnittstellen voraus.

Time-to-Market (◑): Die Architektur unterstützt eine modulare Bauweise von Prozessen, was zu einer potenziellen Beschleunigung der Entwicklungszyklen beiträgt. Dies kann sich positiv auf die Time-to-Market auswirken und somit zu einem schnelleren Rollout neuer Funktionalitäten und geänderten Anforderungen führen. Allerdings hängt die Time-to-Market auch von Faktoren wie der verwendeten Schnittstelle zwischen UiPath und Flowable sowie dem Reifegrad der Entwicklungs- und Deployment-Prozesse ab, da diese ebenfalls eine signifikante Rolle spielen.

Compliance und Governance (◔): In ihrer grundlegenden Form bietet die Architektur keine expliziten Funktionen zur Unterstützung von Compliance und Governance wie Überwachung oder Protokollierung. Sie ist jedoch in der Gestaltung offen für die Integration solcher Mechanismen. Die im Kontext zu Erstellung der Architektur verwendeten Softwarelösungen UiPath und Flowable enthalten bereits eingebaute Governance- und Compliance-Funktionen, welche die Einhaltung betrieblicher Richtlinien und Standards unterstützen.

Administrierbarkeit (◕): Die Architektur zeichnet sich durch ihre klare Strukturierung und Modularität aus, was die Administration erleichtert. Durch die Verwendung von etablierten Tools wie UiPath und Flowable, die bereits über umfassende Verwaltungsoberflächen verfügen, wird die Administrierbarkeit gestärkt. Dennoch hängt eine effektive Administrierbarkeit von der Einrichtung entsprechen der Management- und Monitoring-Tools ab, die in der Lage sind, das System im operativen Betrieb zu unterstützen.

Sicherheit (◔): Die Sicherheit ist ein kritischer Aspekt der Architektur, welcher besondere Aufmerksamkeit erfordert. Durch die Verwendung modularer Komponenten können Sicherheitsmaßnahmen gezielt und spezifisch für jeden Microservice implementiert werden. Dies fördert eine Sicherheitsarchitektur, die sich an den individuellen Sicherheitsanforderungen jeder Komponente orientiert. Die aktuelle Architektur besitzt durch die verwendeten Systeme bereits Sicherheitsmechanismen für Authentifizierung, Autorisierung und Verschlüsselung. Allerdings besteht erhebliches Optimierungspotenzial, insbesondere wenn man im RPA-Bereich Systeme verwendet, die umfassende Rollen- und Berechtigungskonzepte erfordern. Es sollten zusätzliche Maßnahmen ergriffen werden, um die Sicherheit weiter zu verbessern und sicherzustellen, dass keine übermäßigen Rechte existieren, speziell bei der Automatisierung kritischer Systeme.

FAZIT

Der Artikel hat sich mit der Entwicklung einer Microservice-Architektur im RPA-Umfeld beschäftigt, um Vorteile wie bessere Wiederverwendbarkeit, erhöhte Skalierbarkeit, größere Ersetzbarkeit und verbesserte Flexibilität zu realisieren. Diese Aspekte tragen dazu bei, Entwicklungs- und Wartungszeiten zu reduzieren und somit die Agilität und Kosteneffizienz zu steigern.

Ein zentrales Ergebnis ist der Aufbau einer modularen Architektur, die durch die Integration von RPA-Microservices und ein Orchestrierungstool unterstützt wird. Die Katalogisierung der Microservices sowie die Unterscheidung zwischen wertgenerierenden und nicht-wertgenerierenden Services fördern die Übersicht und Wiederverwendbarkeit.

Handlungsempfehlung

Um die langfristige Effizienz und Skalierbarkeit der Architektur sicherzustellen, sollte diese regelmäßig evaluiert und an sich ändernde Anforderungen angepasst werden. Logging- und Monitoring-Konzepte sind ebenfalls weiterzuentwickeln, insbesondere wenn die Anzahl an Microservices steigt. Ein weiterer Aspekt ist die kontinuierliche Überprüfung der eingesetzten Microservices, um unnötige Prozesse zu identifizieren und zusammenzuführen. Workshops und Schulungen sollten regelmä-

ßig angeboten werden, um das Wissen über Microservices im Unternehmen zu vertiefen. Die Optimierung der Schnittstellen zwischen den einzelnen Schichten, möglicherweise durch den Einsatz eines API-Gateways, ist ein zentraler Aspekt, um die Effizienz weiter zu steigern.

Ausblick

Die vorgestellte Architektur bildet die Grundlage für zukünftige Weiterentwicklungen. Durch die Einführung moderner Technologien wie künstlicher Intelligenz könnten Ausfälle proaktiv verhindert und Sicherheitsmaßnahmen verbessert werden. Langfristig könnte die Monetarisierung der entwickelten RPA-Microservices als Umsatzquelle dienen und Unternehmen als Innovationsführer in der Branche positionieren.

Literaturverzeichnis

Aguirre, Santiago; Rodriguez, Alejandro (2017): Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study, S. 65–71. DOI: 10.1007/978-3-319-66963-2_7.

Alberth, Markus; Mattern, Michael (2017): Automation. Understanding robotic process Automation (RPA). In: *JOURNAL - THE CAPCO INSTITUTE JOURNAL OF FINANCIAL TRANSFORMATION* (46). Online verfügbar unter https://www.capco.com/-/media/CapcoMedia/Capco-2023/Capco-Institute/Journal-46/JOURNAL46_5_Alberth.ashx.

Alpers, Sascha; Becker, Christoph; Oberweis, Andreas; Schuster, Thomas (2015): Microservice Based Tool Support for Business Process Modelling. In: *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop,* S. 71–78. DOI: 10.1109/EDOCW.2015.32

Al-Slais, Yaqoob; Ali, Mazen (2023): Robotic Process Automation and Intelligent Automation Security Challenges: A Review. In: *2023 International Conference On Cyber Management And Engineering (CyMaEn)*, S. 71–77. DOI: 10.1109/CyMaEn57228.2023.10050996.

Asatiani, Aleksandre; Penttinen Esko (2016): TURNING ROBOTIC PROCESS AUTOMATION INTO COMMERCIAL SUCCESS – CASE OPUSCAPITA. In: *Journal of Information Technology Teaching Cases* (6(2)), S. 67–74.

AWS (o. D.): Was ist eine API? – Anwendungsprogrammierschnittstelle? Hg. v. Amazon Web Services, Inc. Online verfügbar unter https://aws.amazon.com/de/what-is/api/, zuletzt geprüft am 11.01.2024.

Axmann, Bernhard; Harmoko, Harmoko (2020): Robotic Process Automation: An Overview and Comparison to Other Technology in Industry 4.0. In: *10th International Conference on Advanced Computer Information Technologies (ACIT)*, S. 559–562. DOI: 10.1109/ACIT49673.2020.9208907.

Baškarada, Saša; Nguyen, Vivian; Koronios, Andy (2020): Architecting Microservices: Practical Opportunities and Challenges. In: *Journal of Computer Information Systems* 60 (5), S. 428–436. DOI: 10.1080/08874417.2018.1520056.

Berruti, Federico; Nixon, Graeme; Taglioni, Giambattista; Whiteman, Rob (2017): Intelligent process automation: The engine at the core of the next-generation operating model. In: *McKinsey & Company*, 2017. Online verfügbar unter https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/intelligent-process-automation-the-engine-at-the-core-of-the-next-generation-operating-model, zuletzt geprüft am 11.01.2024.

Blinowski, Grzegorz; Ojdowska, Anna; Przybylek, Adam (2022): Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. In: *IEEE Access* 10, S. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803.

Brettschneider, Jennifer (2020): Bewertung der Einsatzpotenziale und Risiken von Robotic Process Automation. In: *HMD* 57 (6), S. 1097–1110. DOI: 10.1365/s40702-020-00621-y.

Bygstad, Bendik (2017): Generative Innovation: A Comparison of Lightweight and Heavyweight IT. In: *Journal of Information Technology* 32 (2), S. 180–193. DOI: 10.1057/jit.2016.15.

Celonis (o. D.): Unser Unternehmen. Hg. v. Celonis. Online verfügbar unter https://www.celonis.com/de/company/, zuletzt geprüft am 10.03.2024.

Choi, Daehyoun; R'bigui, Hind; Cho, Chiwoon (2021): Candidate Digital Tasks Selection Methodology for Automation with Robotic Process Automation. In: *Sustainability* 13 (16), S. 8980. DOI: 10.3390/su13168980.

Choi, Daehyoun; R'bigui, Hind; Cho, Chiwoon (2022): Enabling the Gab Between RPA and Process Mining: User Interface Interactions Recorder. In: *IEEE Access* 10, S. 39604–39612. DOI: 10.1109/ACCESS.2022.3165797.

Da Costa, Diogo António Silva; Mamede, Henrique São; Da Mira Silva, Miguel (2022): Robotic Process Automation (RPA) Adoption: A Systematic Literature Review. In: *Engineering Management in Production and Services* 14 (2), S. 1–12. DOI: 10.2478/emj-2022-0012.

Dey, Sourav; Das, Arindam (2019): Robotic process automation: assessment of the technology for transformation of business processes. In: *IJBPIM* 9 (3), Artikel 100927, S. 220–230. DOI: 10.1504/IJBPIM.2019.100927.

Doguc, Ozge (2020): Robot Process Automation (RPA) and Its Future. In: Ümit Hacıoğlu (Hg.): Handbook of research on strategic fit and design in business ecosystems. Hershey, PA, USA: IGI Global Business Science Reference (Advances in E-Business Research (AEBR) book series), S. 469–492. Online verfügbar unter https://www.researchgate.net/profile/Ozge-Doguc-2/publication/338302068_Robot_Process_Automation_RPA_and_Its_Future/links/5f5772f592851c250b9d23ad/Robot-Process-Automation-RPA-and-Its-Future.pdf, zuletzt geprüft am 10.12.2023.

Dowalil, Herbert (2018): Grundlagen des modularen Softwareentwurfs. Der Bau langlebiger Mikro- und Makro-Architekturen wie Microservices und SOA 2.0. München: Hanser.

Drawehn, Jens; Krause, Tobias; Renner, Thomas; Kintz, Maximilien (2022): Robotic Process Automation in Versicherungsunternehmen. Erfahrungen und Best Practices beim Einsatz von RPA: Fraunhofer-Gesellschaft. Online verfügbar unter https://www.digital.iao.fraunhofer.de/content/dam/iao/ikt/de/documents/RPA_in_Versicherungsunternehmen.pdf, zuletzt geprüft am 13.12.2023.

Eyerman, Stijn; Hur, Ibrahim (2022): Efficient Asynchronous RPC Calls for Microservices: DeathStarBench Study. DOI: 10.48550/arXiv.2209.13265.

Farley, David (o. D.): Modernes Software Engineering - Bessere Software schneller und effektiver entwickeln by David Farley. Hg. v. O'Reilly. Online verfügbar unter https://www.oreilly.com/library/view/modernes-software-engineering/9783747506363/Text/k10.html, zuletzt geprüft am 20.01.2024.

Fowler, Martin (2020): Domain Driven Design. Hg. v. martinFowler.com. Online verfügbar unter https://martinfowler.com/bliki/DomainDrivenDesign.html, zuletzt geprüft am 15.04.2024.

Fowler, Martin; Lewis, James (2014): Microservices. a definition of this new architectural term. Hg. v. martinFowler.com. Online verfügbar unter https://martinfowler.com/articles/microservices.html?source=post_page, zuletzt aktualisiert am 25.03.2014, zuletzt geprüft am 16.11.2023.

Gartner (o. D.a): Definition of Architecture - Gartner Information Technology Glossary. Hg. v. Gartner. Online verfügbar unter https://www.gartner.com/en/information-technology/glossary/architecture#:~:text=IT%20architecture%20is%20a%20series%20of%20principles%2C%20guidelines,communications%2C%20development%20methodologies%2C%20modeling%20tools%20and%20organizational%20structures., zuletzt geprüft am 19.01.2024.

Gartner (o. D.b): Definition of Robotic Process Automation. Hg. v. Gartner. Online verfügbar unter https://www.gartner.com/en/information-technology/glossary/robotic-process-automation-software, zuletzt aktualisiert am 11.12.2023, zuletzt geprüft am 11.12.2023.

GitLab (2022): What are the benefits of a microservices architecture? Hg. v. GitLab. Online verfügbar unter https://about.gitlab.com/blog/2022/09/29/what-are-he-benefits-of-a-microservices-architecture/, zu- letzt aktualisiert am 29.09.2022, zuletzt geprüft am 23.01.2024.

Habibullah, Safa; Liu, Xiaodong; Tan, Zhiyuan; Zhang, Yonghong; Liu, Qi (2019): Reviving Legacy Enterprise Systems with Micro service-Based Architecture with in Cloud Environments. In: *8th International Conference on Soft Computing, Artificial Intelligence and Applications* 9, S. 173–186. DOI: 10.5121/csit.2019.90713.

Hanussek, Marc (2019): RPA meets KI oder: wie intelligente Softwareroboter Ihre Prozesse automatisieren. Hg. v. Fraunhofer IAO - BLOG. Online verfügbar unter https://blog.iao.fraunhofer.de/rpa-meets-ki-oder-wie-intelligente-softwareroboter-ihre-prozesse-automatisieren/, zuletzt aktualisiert am 06.07.2021, zuletzt geprüft am 28.12.2023.

Hu, Chenglie (2023): An Introduction to Software Design. Concepts, Principles, Methodologies, and Techniques. 1st ed. 2023. Cham: Springer International Publishing; Imprint Springer. Online verfügbar unter https://link.springer.com/book/10.1007/978-3-031-28311-6.

IEEE Computer Society/Software & Systems Engineering Standards (2022): IEEE/ISO/IEC International Standard for Software, systems and enterprise--Architecture description. DOI: 10.1109/IEEESTD.2022.9938446.

Indrasiri, Kasun; Siriwardena, Prabath (2018): Microservices for the Enterprise. Designing, Developing, and Deploying. 1st ed. 2018. New York: Apress (Springer eBook Collection). Online verfügbar unter https://link.springer.com/content/pdf/10.1007/978-1-4842-3858-5.pdf.

Institute for Robotic Process Automation & Artificial Intelligence (o. D.): What is Robotic Process Automation? | IRPAAI. Hg. v. IRPA AI. Online verfügbar unter https://irpaai.com/what-is-robotic-process-automation/, zuletzt aktualisiert am 11.12.2023, zuletzt geprüft am 11.12.2023.

Ivančić, Lucija; Suša Vugec, Dalia; Bosilj Vukšić, Vesna (2019): Robotic Process Automation: Systematic Literature Review 361, S. 280–295. DOI: 10.1007/978-3-030-30429-4_19.

Jamshidi, Pooyan; Pahl, Claus; Mendonca, Nabor C.; Lewis, James; Tilkov, Stefan (2018): Microservices: The Journey So Far and Challenges Ahead. In: IEEE Softw. 35 (3), S. 24–35. DOI: 10.1109/MS.2018.2141039.

Kalske, Miika; Mäkitalo, Niko; Mikkonen, Tommi (2018): Challenges When Moving from Monolith to Microservice Architecture. In: Irene Garrigós und Manuel Wimmer (Hg.): Current Trends in Web Engineering. ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine ; Rome, Italy, June 5-8, 2017 ; revised selected papers, Bd. 10544. Cham: Springer International Publishing (Lecture Notes in Computer Science, 10544), S. 32–47. Online verfügbar unter https://link.springer.com/chapter/10.1007/978-3-319-74433-9_3.

Karnowski, Veronika (2013): Diffusionstheorie. In: Wolfgang Schweiger und Andreas Fahr (Hg.): Handbuch Medienwirkungsforschung. Wiesbaden: Springer VS, S. 513–528. Online verfügbar unter https://link.springer.com/chapter/10.1007/978-3-531-18967-3_27.

Khan, Ovais; Siddiqui, Nabil; Oleson, Timothy; Fussell, Mark (2021): Embracing Microservices Design. A practical guide to revealing anti-patterns and architectural pitfalls to avoid microservices fallacies. 1st edition. Erscheinungsort nicht ermittelbar, Boston, MA: Packt Publishing; Safari.

Kirchmer, Mathias (2017): Robotic Process Automation - Pragmatic Solution or Dangerous Illusion? In: *BTOES Insights (Business Transformation and Operational Excellence Summit Insights)*. Online verfügbar unter https://www.researchgate.net/publication/317730848_Robotic_Process_Automation_-_Pragmatic_Solution_or_Dangerous_Illusion, zuletzt geprüft am 04.01.2024.

Koch, Oliver; Wildner, Stephan (2020): Intelligent Robotic Process Automation. Konzeption eines Ordnungsrahmens zur Nutzung künstlicher Intelligenz für die Prozessautomatisierung. In: Rüdiger Buchkremer, Thomas Heupel und Oliver Koch (Hg.): Künstliche Intelligenz in Wirtschaft & Gesellschaft. Auswirkungen, Herausforderungen & Handlungsempfehlungen. Wiesbaden, Heidelberg: Springer Gabler (FOM-Edition), S. 211–230. Online verfügbar unter https://link.springer.com/book/10.1007/978-3-658-29550-9, zuletzt geprüft am 04.01.2024.

Köhler-Schute, Christiana (2020): Robotic Process Automation in Unternehmen. Praxisorientierte Methoden und Vorgehensweisen zur Umsetzung von RPA-Initiativen. Berlin: KS-Energy-Verlag.

Kokina, Julia; Blanchette, Shay (2019): Early evidence of digital labor in accounting: Innovation with Robotic Process Automation. In: *International Journal of Accounting Information Systems* 35, S. 100431. DOI: 10.1016/j.accinf.2019.100431.

Kroll, Christian; Bujak, Adam; Darius, Volker; Enders, Wolfgang; Esser, Marcus (2016): Robotic Process Automation - Robots conquer business processes in back offices. A 2016 study conducted by Capgemini Consulting and Capgemini Business Services. Hg. v. Capgemini. Online verfügbar unter https://www.capgemini.com/consulting-de/wp-content/uploads/sites/32/2017/08/robotic-process-automation-study.pdf, zuletzt geprüft am 04.01.2023.

Lal Sahni, Dhanik (2023): What is Microservice Architecture? Hg. v. Salesforcecodex. Online verfügbar unter https://stories.salesforcecodex.com/2023/05/salesforce/what-is-microservice-architecture/, zuletzt aktualisiert am 17.05.2023, zuletzt geprüft am 19.11.2023.

Langmann, Christian; Turi, Daniel (2021): Robotic Process Automation (RPA) - Digitalisierung und Automatisierung von Prozessen. Voraussetzungen, Funktionsweise und Implementierung am Beispiel des Controllings und Rechnungswesens. 2. Auflage. Wiesbaden, Heidelberg: Springer Gabler.

Lhuer, Xavier (2016): The next acronym you need to know about: RPA (robotic process automation. Hg. v. McKinsey. Online verfügbar unter https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-next-acronym-you-need-to-know-about-rpa, zuletzt geprüft am 13.05.2024.

Madakam, Somayya; Holmukhe, Rajesh M.; Kumar Jaiswal, Durgesh (2019): The Future Digital Work Force: Robotic Process Automation (RPA). In: *JISTEM* 16, S. 1–17. DOI: 10.4301/S1807-1775201916001.

Manning, Louise (2020): Moving from a compliance-based to an integrity-based organizational climate in the food supply chain. In: *Comprehensive reviews in food science and food safety* 19 (3). DOI: 10.1111/1541-4337.12548.

Microsoft Learn (o. D.): Microservice-Architekturstil. Hg. v. Microsoft Learn. Online verfügbar unter https://learn.microsoft.com/de-de/azure/architecture/guide/architecture-styles/microservices, zuletzt aktualisiert am 19.11.2023, zuletzt geprüft am 19.11.2023.

Microsoft Learn (2023): Entwerfen einer an Microservice orientierten Anwendung. Hg. v. Microsoft Learn. Online verfügbar unter https://learn.microsoft.com/de-de/dotnet/architecture/microservices/multi-container-microservice-net-applications/microservice-application-design, zuletzt aktualisiert am 10.05.2023, zuletzt geprüft am 19.11.2023.

Moffitt, Kevin C.; Rozario, Andrea M.; Vasarhelyi, Miklos A. (2018): Robotic Process Automation for Auditing. In: *Journal of Emerging Technologies in Accounting* 15 (1), S. 1–10. DOI: 10.2308/jeta-10589.

Montemagno, James; Warren, Genevieve; Jain, Tarun; Coulter, David; Veloso, Miguel et al. (2022): Communication in a microservice architecture. Hg. v. Microsoft Learn. Online verfügbar unter https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-contai-nerapplications/communication-in-microservicearchitecture, zuletzt aktualisiert am 21.01.2024, zuletzt geprüft am 21.01.2024.

Nadareishvili, Irakli; Mitra, Ronnie; McLarty, Matt; Amundsen, Michael (2016): Microservice architecture. Aligning principles, practices, and culture. First Edition, Second Release. Beijing, Boston, Farnham, Sebastopol, Tokyo: O´Reilly.

Newman, Sam (2020): Vom Monolithen zu Microservices. Patterns, um bestehende Systeme Schritt für Schritt umzugestalten. Heidelberg: O'Reilly.

Newman, Sam (2021): Building microservices. Designing fine-grained systems. Second edition. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly Media.

Niedermaier, Sina; Koetter, Falko; Freymann, Andreas; Wagner, Stefan (2019): On Observability and Monitoring of Distributed Systems – An Industry Interview Study. In: Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, Zahir Tari und Pagination_Cover (Hg.): Service-Oriented Computing. 17th International Conference, ICSOC 2019, Toulouse, France, October 28–31, 2019, Proceedings, Bd. 11895. 1st ed. 2019. Cham: Springer (Springer eBooks Computer Science, 11895), S. 36–52. Online verfügbar unter https://link.springer.com/chap- ter/10.1007/978-3-030-33702-5_3.

Průcha, Petr; Skrbek, Jan (2022): API as Method for Improving Robotic Process Automation. In: Andrea Marrella, Raimundas Matulevičius, Renata Gabryelczyk, Bernhard Axmann, Vesna Bosilj Vukšić, Walid Gaaloul et al. (Hg.): Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum. BPM 2022 Blockchain, RPA, and CEE Forum, Münster, Germany, September 11–16, 2022, Proceedings. 1st ed. 2022. Cham: Springer International Publishing; Imprint Springer (Lecture Notes in Business Information Processing, 459). Online verfügbar unter https://link.springer.com/chapter/10.1007/978-3-031-16168-1_17.

PWC South Africa (o. D.): Robotic process automation. Hg. v. PWC South Africa. Online verfügbar unter https://www.pwc.co.za/en/services/consulting/robotic-process-automation.html, zuletzt geprüft am 11.12.2023.

RedHat (2019): Wie funktioniert ein API-Gateway? Hg. v. RedHat. Online verfügbar unter https://www.redhat.com/de/topics/api/what-does-an-api-gateway-do, zuletzt geprüft am 07.12.2023.

Santos, Filipa; Pereira, Rúben; Vasconcelos, José Braga (2020): Toward robotic process automation implementation: an end-to-end perspective. In: *BPMJ* 26 (2), S. 405–420. DOI: 10.1108/bpmj-12-2018-0380.

SAP (o. D.): Was ist Prozessautomatisierung? Hg. v. SAP. Online verfügbar unter https://www.sap.com/germany/products/technology-platform/process-automation/what-is-process-automation.html, zuletzt geprüft am 10.12.2023.

Shidaganti, Ganeshayya; Salil, Sreya; Anand, Prarthana; Jadhav, Vaishnavi (2021): Robotic Process Automation with AI and OCR to Improve Business Process: Review. In: *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, S. 1612–1618. DOI: 10.1109/ICESC51422.2021.9532902.

Singleton, Andy (2016): The Economics of Microservices. In: IEEE Cloud Comput. 3 (5), S. 16–20. DOI: 10.1109/MCC.2016.109.

Syed, Rehan; Suriadi, Suriadi; Adams, Michael; Bandara, Wasana; Leemans, Sander J.J.; Ouyang, Chun et al. (2020): Robotic Process Automation: Contemporary themes and challenges. In: *Computers in Industry* 115, S. 103162. DOI: 10.1016/j.compind.2019.103162.

Taibi, Davide; Lenarduzzi, Valentina; Pahl, Claus (2017): Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. In: *IEEE Cloud Comput.* 4 (5), S. 22–32. DOI: 10.1109/MCC.2017.4250931.

UiPath (o. D.): KI und RPA – die nächste Stufe der Automatisierung | UiPath. Hg. v. UiPath. Online verfügbar unter https://www.uipath.com/de/automation/ai-and-rpa, zuletzt geprüft am 28.12.2023.

van der Aalst, Wil M. P.; Bichler, Martin; Heinzl, Armin (2018): Robotic Process Automation. In: *Bus Inf Syst Eng* 60 (4), S. 269–272. DOI: 10.1007/s12599-018-0542-4.

Vitharanage, Imesha; Thibbotuwawa, Amila (2021): Enterprise Robotic Process Automation. In: *BPRM* 01 (01), S. 10–12. DOI: 10.31705/BPRM.2021.2.

Wanner, Jonas; Hofmann, Adrian; Fischer, Marcus; Janiesch, Christian; Imgrund, Florian; Geyer-Klingebert, Jerome (2019): Process Selection in RPA Projects – Towards a Quantifiable Method of Decision Making. In: *Fortieth International Conference on Information Systems*. Online verfügbar unter https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/deliver/index/docId/95923/file/95923.pdf, zuletzt geprüft am 06.01.2024.

Willcocks, Leslie; Lacity, Mary; Craig, Andrew (2015): The IT Function and Robotic Process Automation. In: *London School of Economics and Political Science*. Online verfügbar unter https://eprints.lse.ac.uk/64519/1/OUWRPS_15_05_published.pdf, zuletzt geprüft am 10.12.2023.

Wolff, Eberhard (2018): Microservices. Grundlagen flexibler Softwarearchitekturen. 2., aktualisierte Auflage. Heidelberg: dpunkt.verlag.

Yousif, Mazin (2016): Microservices. In: *IEEE Cloud Comput.* 3 (5), S. 4–5. DOI: 10.1109/MCC.2016.101.

Zhang, Chanyuan; Thomas, Chanta; Vasarhelyi, Miklos A. (2021): Attended Process Automation in Audit: A Framework and A Demonstration. In: Journal of Information Systems 36 (2). DOI: 10.2308/ISYS-2020-073.