Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik Software Engineering

Integrations- und Bereitstellungsautomatisierung von Cloud-Anwendungen für Composable-Enterprise-Architekturen im Kontext der SAP Business Technology Platform

Bachelorarbeit

Im Rahmen der Prüfung zum Bachelor of Science (B. Sc.)

Verfasser:	Rafael Martin
Kurs:	WI SE-B 2020
Dualer Partner:	SAP SE, Walldorf
Betreuer der Ausbildungsfirma:	Klaus Räwer
Wissenschaftlicher Betreuer:	Herr Ulrich Wolf
Abgabedatum:	08.05.2023

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema:

Integrations- und Bereitstellungsautomatisierung von Cloud-Anwendungen für Composable-Enterprise-Architekturen im Kontext der SAP Business Technology Platform

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, 08.05.2023, _____

Rafael Martin

Inhaltsverzeichnis

Abkürzungsverzeichnis

XP	Extreme Programming
DevOps	Development & Operations
CI/CD	Cuntinuous Integration and Continuous Delivery
CI	Cuntinuous Integration
CD	Cuntinuous Delivery
DoD	Definition of Done
E2E-Tests	End-to-End-Tests
PBC	Packaged-Business-Capability
CEA	Composable-Enterprise-Architekur
MACH	Microservices, APIs, Cloud-native, Headless
CE	Composable-Enterprise
EDA	Event-driven Architecture
NIST	National Institute of Standards and Technology
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
BTP	Business Technology Platform

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

1.2 Zielsetzung und Abgrenzung

1.3 Aufbau der Arbeit

2 Grundlagen und Begriffserklärungen

2.1 Die Composable-Enterprise-Architektur

2.1.1 Begriffserklärung und Abgrenzung

Flexibilität, Resilienz und Agilität. Nach Ansicht von Steve Denning, Managementberater und Autor der Forbes, sind dies Eigenschaften, in welchen sich „erfolgreiche von erfolglosen Unternehmen unterscheiden“. Für Analystenhäusern wie Gartner stet dabei fest, dass es technologischer Innovation benötigt, um einhergehende Herausforderungen erfolgreich zu bewältigen und eine kontinuierliche Unternehmenstransformation voranzutreiben. Gartner empfiehlt dabei monolithische und starre Unternehmensarchitekturen, durch einen modularen Organisationsaufbau zu ersetzen. In seinen Veröffentlichungen verwendet Gartner für dieses Konzept den Begriff der **Composable-Enterprise-Architekur (CEA)**.

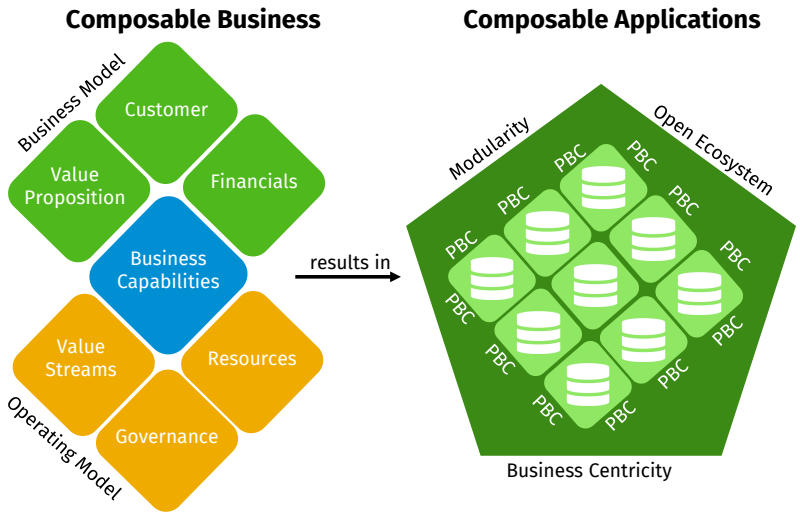

Abbildung 1: Entstehung einer Composable-Enterprise-Architektur. In Anlehnung an Schönstein [22].

In der Literatur wird die CEA dabei wie folgt definiert:

„The Composable-Enterprise-Architecture is an architecture that delivers business outcomes and adapts to the pace of business change. It does this through the assembly and combination of packaged business capabilities [19].“

Ein Composable-Enterprise ist somit ein aus mehreren Bausteinen, sog. *Packaged-Business-Capability (PBC)* bestehendes Unternehmen. PBCs sind vorgefertigte Softwareelemente, welche jeweils eine bestimmte Geschäftsfunktion abdecken (s. Abb. 1). Damit ein Unternehmen zu einem Composable-Enterprise (CE) wird, muss dieses drei Grundsätze einhalten: *Modulare Architektur*, *Offenes Ökosystem* und *Businesszentriertheit* [19]. Laut Gartner müssen Unternehmen nicht nur „akzeptieren, dass der disruptive Wandel zur Normalität gehört". Vielmehr sollten diese den disruptiven Wandel als „Chance begreifen und ihn nutzen, um eine *modulare Architektur* zu implementieren" [19]. Ergibt sich eine Änderung der Geschäftsanforderungen ermöglicht diese Architektur ein flexibles und isoliertes Austauschen, Verändern und Weiterentwickeln einzelner PBCs. Um eine auf den Geschäftszweck maßgeschneiderte IT-Lösung zu implementieren, werden die einzelnen spezialisierten Komponenten kombiniert und verknüpft [3, S. 315]. So kann ein Composable-E-Commerce-Enterprise Elemente, wie den Warenkorb, die Produktsuche oder die Zahlungsabwicklung in modulare Systeme auslagern. Bei Expansion in das Ausland, könnte das Unternehmen eine Komponente zur Abwicklung von Auslandszahlungen integrieren, ohne dabei umfassende Systemänderungen durchführen zu müssen. Das Prinzip des *offenen Ökosystems* ermöglicht Anwendern und Entwicklern Werkzeuge, welche zur Unterstützung der operativen Tätigkeiten benötigen werden, selbst zusammenzustellen und frei zu kombinieren. Die in dem offenen Ökosystem integrierbaren Tools werden dabei auf einem Marktplatz gebündelt und können ohne hohen Aufwand aktiviert und verwendet werden [9, S. 58]. Dazu gehören etwa Tools zur Automatisierung von Prozessen oder Kollaborationsdienste zur Unterstützung der Zusammenarbeit innerhalb Teams. Neben diesen administrativen Werkzeugen werden auf Marktplätzen ebenfalls offene Technologiestandards, wie Datenbanken oder Sicherheitstools angeboten. Diese werden von CEs zur Unterstützung der Anwendungsentwicklung verwendet. Mit der Nutzung solcher externen Standards, Services, und Tools können CEs die eigenen Fähigkeiten im Bereich der nicht-kerngeschäftlichen Kompetenzen erweitern und somit Wettbewerbsvorteile erlangen [11, S. 7]. Um eine anwenderzentrierte Gestaltung der auf dem Marktplatz angebotenen Werkzeuge zu

ermöglichen, sollte der Fokus dieser Tools auf den Bedürfnissen und Erwartungen der Nutzer liegen (*Businesszentriertheit*). IT-Systeme dienen dem Zweck der Unterstützung operativer Aufgaben. Entsprechen diese nicht den Anforderungen der Nutzer, resultiert dies in ineffizienten Arbeitsprozessen. Deshalb ist essenziell, dass Mitarbeiter Systeme intuitiv nutzen und ggf. weiterentwickeln und anpassen können, ohne dabei von der IT-Abteilungen abhängig zu sein.[19]

2.1.2 Technologische Konzepte des Composable-Enterprises

Um die in Kapitel 2.1.1 beschriebenen betriebswirtschaftlichen Grundsätze in das eigene Unternehmen zu integrieren, benötigt es verschiedener technologischer Konzepte. Zusammenfassen lässt sich dieses mit dem Akronym *MACH*: *Microservices, APIs, Cloud-native, Headless.*

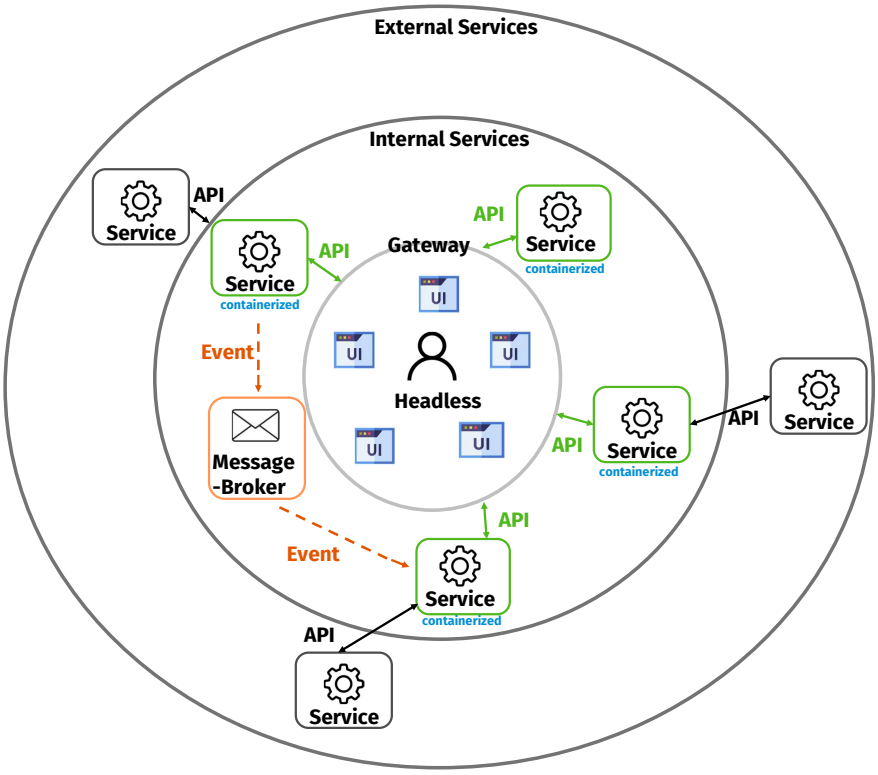

Abbildung 2: Technische Realisierung der Composable-Enterprise-Architektur. Eigene Darstellung.

Der zentrale Einstiegspunkt des Nutzers (Client) in einer CE-Anwendung ist das User-Interface. Dieses kann dabei auf einer Desktop-Anwendung am Computer, einer Website im Browser oder einer App auf dem Handy abgerufen werden. Insbesondere für Content-Management-Systeme wird für Frontend-Entwicklungen das *Headless-Konzept* verwendet. Dieses beschreibt, dass zwischen Front- und Backend keine feste Kopplung besteht. Vielmehr werden auf dem Backend standardisierte Daten verwaltet, welche auf verschiedenen Frontends ausgegeben werden können [19]. Die Applikationslogik wird in kleine isolierte *Microservices* gekapselt. Um Nachrichten zwischen Frontend und den Services zu übermitteln wird i.d.R. ein Gateway zwischengeschaltet [9, S. 41]. Mit diesem wird eine zentrale und standardisierte Schnittstelle bereitgestellt, welche verschiedene Authentifizierungs-, Authorisierungs- sowie Routing-Mechanismen implementiert. Die einzelnen Services werden auf losen Instanzen betrieben, welche jedoch im Zusammenspiel eine große Anwendung darstellen. Dabei ermöglicht sich, für jeden Service eine unterschiedliche Programmiersprache sowie Datenbank zu verwenden. So können Architektur und Technologien eines Services unmittelbar an dessen betriebswirtschaftliche Anforderungen angepasst werden. Zur Kommunikation zwischen Services werden standardisierte Schnittstellen, sog. *Application-Programming-Interfaces (APIs)* verwendet. Mit APIs werden die von den Services bereitgestellten Funktionalitäten und Daten veröffentlicht [1, S. 15]. Diese Schnittstellen können dabei unmittelbar von dem Frontend oder anderen Microservices konsumiert werden. Zur Implementierung von APIs werden dabei i.d.R. Protokolle wie HTTP oder OData verwendet. Ein weiteres bei CEs verwendetes Kommunikationskonzept ist die *Event-driven Architecture (EDA)*. Während APIs auf konkrete Serviceanfrage wie z.B. HTTP-GET-Requests reagieren, bezweckt die EDA eine asynchrone Verarbeitung von Events. Dabei werden einzelnen Services Rollen eines *Event Producers* bzw. *Event Consumers* zugewiesen [2, S. 51]. Der Event Producer erzeugt Nachrichten und übermittelt diese einer unabhängigen Instanz zur Verwaltung der Events (*Message Broker*) [2, S. 61]. Ein Event Consumer kann dabei bestimmte Themen des Message Brokers abonnieren und gesendete Ereignisse auslesen bzw. verarbeiten [2, S. 54]. Da der Message Broker eine unabhängige Vermittlungsinstanz zwischen den Services darstellt, können neue Diens-

te Event-Themen abonnieren und somit ohne hohen Aufwand in eine bestehende Kommunikation eingegliedert werden. Alle Komponenten der CEA werden auf einer Cloud-Plattform betrieben (*cloud-native*). Cloud-Computing ist ein Dienstleistungsmodell, welches Nutzern ermöglicht Ressourcen, wie Speicher, Analyse-Tools oder Software über das Internet von einem Cloud-Anbieter zu beziehen [10, S. 5]. Für das Cloud-Computing werden durch das National Institute of Standards and Technology (NIST) verschiedene Servicemodelle definiert. Neben *Software-as-a-Service (SaaS)* und *Infrastructure-as-a-Service (IaaS)*, bei welchem eine Anwendung bzw. eine gesamte Infrastruktur in der Cloud gemietet wird, gibt es ebenfalls das *Platform-as-a-Service (PaaS)* [10] [10, S. 9]. Bei diesem Computing-Modell wird eine Plattform bereitgestellt, auf welcher Kunden eigene Anwendungen entwickeln, testen und betreiben können. Ein auf dieser Service-Ebene von der SAP bereitgestelltes Produkt ist die SAP Business Technology Platform (SAP BTP). Diese stellt eine Reihe von Diensten und Funktionen zur Verfügung, mit welchen Kunden die eigenen SAP-ERP-Systeme anpassen, integrieren und erweitern können. PaaS ermöglicht IT-Services schnell und kosteneffizient an aktuelle Markterfordernisse anzupassen. Aufgrund der nutzungsabhängigen Bepreisung von Cloud-Plattformen können Dienste ohne hohen Investitionseinsatz auf- und abgebaut werden. Da Services in Cloud-Plattformen i.d.R. auf virtuellen bzw. containerisierten Umgebungen betrieben werden, können Anwendungen schnell und effizient skaliert und somit stets die benötigte Rechenleistung bereitgestellt werden [10, S. 10].

2.2 Integration und Bereitstellung von Software

2.2.1 Agile und DevOps als moderne Anwendungsentwicklungskonzepte

Das Hauptaugenmerk eines CEs besteht darin eine möglichst modulare und flexible Systemarchitektur zu schaffen. Damit soll sichergestellt werden, dass IT-Leistungen in einem sich stetig ändernden Umfeld schnell und risikoarm bereitgestellt werden. Das traditionelle Wasserfallmodell, welches eine sequenzielle Abfolge der Projektelemente *Anforderung*, *Design*, *Implementierung*, *Test* und *Betrieb* vorgibt, besitzt dabei signifikante Limitationen. Die in dieser Methodik detailliert durchgeführte Vorabplanung, kann insbesondere in umfangreichen Langzeitvorhaben aufgrund unvorhersehbarer Externalitäten selten eingehalten werden. Auch die starre Abfolge der Projektphasen mindert insbesondere in fortgeschrittenen Zeitpunkten des Vorhabens den Spielraum für Anpassungsmöglichkeiten [14, S. 5]. Dies resultiert nicht nur einem Anstieg der Kosten, sondern führt ebenfalls dazu, dass IT-Projekte länger als geplant ausfallen [13, S. 41]. Als Reaktion haben sich innerhalb der Projektmanagementlandschaft zunehmend **agile Vorgehensmodelle** etabliert. Im Gegensatz zum Wasserfallmodell, welches eine umfassende Vorabplanung vorsieht, wird das Vorhaben in einer agilen Entwicklung in viele zyklische Einheiten, sog. *Sprints*, segmentiert (s. Abb. 3) [4, S. 87]. Alle innerhalb des Projektumfangs zu entwickelnden Funktionalitäten werden dabei in einem zentralen Artefakt (*Product Backlog*) festgehalten und von dem Produktverantwortlichen (*Product Owner*) priorisiert.

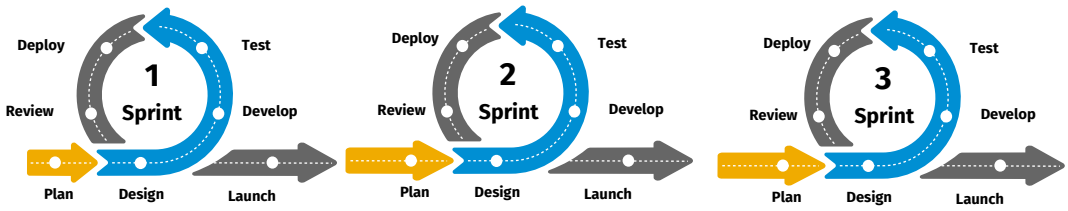

Abbildung 3: Exemplarische Abfolge eines agilen Entwicklungszykluses. In Anlehnung an K&C [16].

Sprints sind Durchläufe, welche i.d.R. einen Zeitraum von ein bis vier Wochen umfassen. Während dieses Abschnitts ist die Fertigstellung einer vor dem Sprint definierten Aufgabenkontingente (*Sprint Backlog*) vorgesehen. Nach Abschluss eines Sprints

soll dabei ein potenziell an den Kunden auslieferbares Produkt zur Verfügung gestellt werden. Dies erlaubt eine schnelle Bereitstellung funktionsfähiger Software, was neben einem beschleunigtem Kundennutzen ebenfalls in einer Optimierung der Planungsprozesse resultiert. So kann das nach Ablauf eines Sprints an die Stakeholder ausgelieferte Artefakt als Feedback-Grundlage verwendet und im unmittelbaren Folge-Sprint eingearbeitet werden [16, S. 39]. Innerhalb der letzten Dekade haben sich diverse auf agilen Prinzipien basierenden Vorgehensmodelle, wie Scrum, Kanban oder Extreme Programming (XP) in der Softwareentwicklung etabliert. Obwohl einige dieser Methoden zur erfolgreichen Zusammenarbeit innerhalb der Entwicklungsteams beigetragen haben, bleibt das sog. *Problem der letzten Meile* bestehen [17]. Traditionell erfolgt eine funktionale Trennung der Entwickler- und IT-Betriebler-Teams. Das Problem der letzten Meile beschreibt dabei, dass aufgrund ausbleibender Kooperation der Entwicklungs- und Betriebsteams der Programmcode nicht auf die Produktivumgebung abgestimmt ist. Erkenntnisse aus der Praxis zeigen, dass solche organisatorischen Silos häufig in einer schlechten Softwarequalität und somit in einem geminderten Ertragspotenzial bzw. in einer Erhöhung der Betriebskosten resultieren [5, S. 1]. So geht aus der von McKinsey veröffentlichen Studie *The Business Value of Design 2019* hervor, dass durchschnittlich 80 Prozent des Unternehmens-IT-Budgets zur Erhaltung des Status quo, also zum Betrieb bestehender Anwendungen verwendet wird. Stattdessen fordert das Beratungshaus eine Rationalisierung der Bereitstellung von Software, um finanzielle Mittel für wertschöpfende Investitionen zu maximieren [20]. Abhilfe schaffen kann das in der Literatur als **Development & Operations (DevOps)** bekannte Aufbrechen organisatorischer Silos zwischen Entwicklung und dem IT-Betrieb [5, S. 1]. Dabei stellt DevOps keine neue Erfindung dar. Stattdessen werden einzelne bereits bewährte Werkzeuge, Praktiken und Methoden, wie z.B. die agile Softwareentwicklung, zu einem umfassenden Rahmenwerk konsolidiert. DevOps zielt dabei auf eine Optimierung des gesamten Applikationslebenszykluses, von Planung bis Bereitstellung der Software, ab. Neben der Bereitstellung von IT-Services umfasst DevOps ebenfalls Aktivitäten zu IT-Sicherheit, Compliance und Risikomanagement. Prägnant zusammenfassen lässt sich das DevOps-Konzept

durch das Akronym CAMS: *Culture (Kultur)*, *Automation (Automatisierung)*, *Measurement (Messung)* und *Sharing (Teilen)* [5, S. 5]. Dabei gilt *Kultur* als das wohl wesentlichste DevOps-Erfolgselement. Diese bezweckt eine Kollaborationsmentalität, welche sich über alle Ebenen eines Unternehmens erstreckt. Operative Entscheidungen sollen dabei auf die Fachebenen herunter delegiert werden, welche aufgrund ihrer spezifischen Expertise am geeigneten sind, Dispositionen zu verabschieden [5, S. 5]. Eine *Automatisierung* der Softwarebereitstellungsprozesse ermöglicht, sich wiederholende manuelle Arbeit zu eliminieren. Dies kann ebenfalls zur Rationalisierung und damit zur Senkung der IT-Betriebskosten beitragen. Der dabei erzielte Einfluss wird anhand verschiedener DevOps-Kennzahlen bemessen (*Messung*). Neben der Systemverfügbarkeit und der Instandsetzungszeit sind für Softwareentwicklungsunternehmen insbesondere *Time-to-Market* sowie *Time-to-Value* signifikante Metriken [5, S. 7].

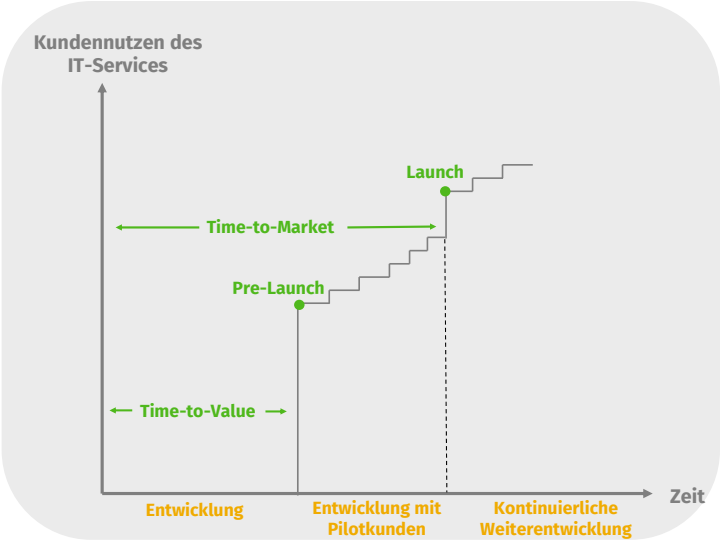

Abbildung 4: Zeitliche Darstellung der Herbeiführung von Kundennutzen bei der Entwicklung von IT-Services [5, S. 9].

Der Time-to-Market beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [12, S. 141]. Auch der *Time-to-Value* erhält zunehmend Bedeutung in der Softwareentwicklung. Im Gegensatz zum Time-to-Market wird hier nicht die Zeit bis zur Komplett-Einführung, sondern das Intervall bis die von dem Softwareunternehmen entwickelte Lösung ersten

Kundennutzen herbeiführt, bemessen. Obwohl der im Time-to-Value bereitgestellte IT-Service möglicherweise Verbesserungspotenzial besitzt, überwiegt für Kunden des Unternehmens der mit der initialen Auslieferung herbeigeführte Mehrwert. Eine solche Früheinführung ermöglicht dem Softwareunternehmen ebenfalls einen Vorsprung gegenüber Konkurrenten. So ist diesem bereits gelungen, erste Kunden zu akquirieren, deren Input und Feedback möglichst rasch erfasst und verarbeitet werden kann [5, S. 9]. Softwareunternehmen können IT-Services ab dem Pre-Launch somit sukzessive und ressourcenoptimiert unter Zusammenarbeit mit den Pilotkunden erweitern. Auch Adam Caplan, leitender Strategieberater bei Salesforce, empfiehlt angesichts der bei Softwareintegration entstehenden Komplexität, Anwendungen schnellst möglichst in produktionsähnlichen Umgebungen zu testen [12]. Aus diesen Erfahrungen sollen Best-Practises entwickelt werden, welche innerhalb von Teams und organisationsübergreifend weitergegeben werden (*Teilen*) [5, S. 7].

2.2.2 Pipelines zur Integrations- und Bereitstellungsautomatisierung

DevOps beschreibt eine Philosophie zur Förderung der Zusammenarbeit zwischen Entwicklungs- und Betriebsteams. Ein integraler Bestandteil des DevOps-Rahmenwerks ist *Cuntinuous Integration and Continuous Delivery (CI/CD)*. CI/CD ist ein Verfahren, welches zur Verbesserung der Qualität bzw. zur Senkung der Entwicklungszeit von IT-Services beiträgt. Abhilfe schaffen soll dabei eine Pipeline, welche alle Schritte von Code-Integration bis Bereitstellung der Software automatisiert. Hauptaugenmerk liegt dabei auf einer zuverlässigen und kontinuierlichen Bereitstellung von Software [15, S. 471].

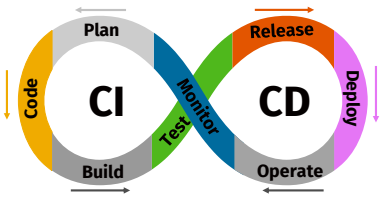

Abbildung 5: Aktivitäten im CI/CD-Prozess. In Anlehnung an Synopsys [23].

Alle in diesem Prozess anfallenden Aktivitäten sind dabei dem CI/CD-Zyklus zu entnehmen (s. Abb. 5). Der CI-Prozess (Continuous-Integration-Prozess) bezweckt, dass lokale Quellcodeänderungen in kurzen Intervallen und so schnell wie möglich in eine zentrale Codebasis geladen werden. Das frühzeitige Integrieren von Code soll dabei zu einer unmittelbaren und zuverlässigen Fehlererkennung innerhalb des Entwicklungsvorhabens beitragen [15, S. 471]. Der erste Schritt des CI-Prozesses umfasst die Planung zu entwickelnder Services (*Plan*: s. Abb. 5). Dabei soll festgestellt werden, welche Anforderungen eine Lösung besitzt bzw. welche Softwarearchitekturen sowie Sicherheitsmaßnahmen implementiert werden sollten. Um sicherzustellen, dass die in der Planung entworfene Anwendungsarchitektur auf das Design des Produktivsystems abgestimmt ist, sollte zu jedem Zeitpunkt das Know-how der Betriebsteams einbezogen werden [5, S. 16]. Nach erfolgreichem Entwurf zu implementierender Anwendungsfeatures beginnt die Entwicklung der IT-Services (*Code*: s. Abb. 5). Arbeiten hierbei mehrere Entwickler parallel an demselben IT-Service, wird der entsprechende Quellcode in Versionsverwaltungssysteme (*Repositorys*) wie Github oder Bitbucket ausgelagert. Ein Repository stellt dabei einen zentralen Speicherort dar, welcher das Verfolgen sowie Überprüfen von Änderungen und ein paralleles bzw. konkurrierendes Arbeiten an einer gemeinsamen Codebasis ermöglicht [8, S. 31]. Der in dem Repository archivierte Hauptzweig (*Master-Branch*) stellt dabei eine aktuelle und funktionsfähige Version des Codes dar. Dieser mit verschiedenen Validierungsprozessen überprüfte Code, stellt dabei die aktuelle in dem Produktionssystem laufende Anwendungsversion dar (s. Abb. 6). Im Sinne der agilen Entwicklung werden dabei große Softwareanforderungen (*Epics*), in kleine Funktionalitäten (*User Storys*) segmentiert, welche in separate Feature-Branches ausgelagert werden. Diese sind unabhängige Kopien des Hauptzweiges, in welcher ein Entwickler Änderungen vornehmen kann, ohne Konflikte in der gemeinsamen Codebasis zu verursachen. Nach Fertigstellung der Funktionalitäten sollte der um die Features erweiterte Quellcode so schnell wie möglich in den Hauptzweig integriert werden. Da mit dieser Zusammenführungen automatische Validierungsprozesse ausgelöst werden, kann mit erfolgreicher Absolvierung der Tests sichergestellt, dass der Code stets stabil, also funktionsfähig ist und keine Konflikte mit dem aktuellen Code

des Hauptzweiges aufweist [8, S. 169]. Die in diesem Schritt abgewickelten Tests leiten sich dabei aus der *Definition of Done (DoD)* ab. Die DoD ist eine in der Planungsphase festgelegte Anforderungsspezifikation, deren Erfüllung als notwendige Voraussetzung für den Abschluss eines Features gilt. Somit sind Entwickler dazu angehalten, für jedes implementierte Feature einen der DoD entsprechenden Test zu entwerfen.

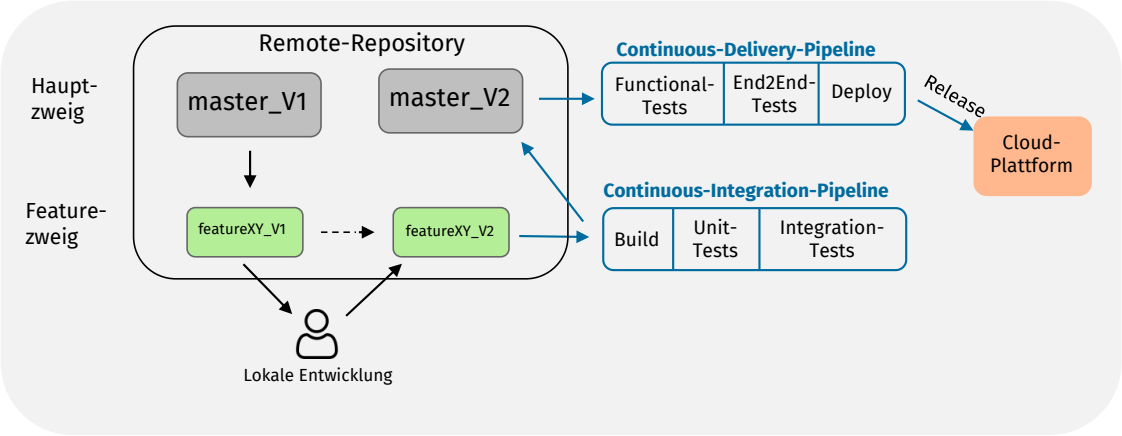

Abbildung 6: Versionskontrollsysteme zur Verwaltung von Quellcode.
Eigene Darstellung.

Die Einbindung des Feature-Branches in den Hauptzweig resultiert i.d.R. in einem unmittelbaren Start des *CI/CD-Pipeline-Prozesses*. Bei der CI/CD-Pipeline handelt es sich dabei um eine vom Repository unabhängige Recheninstanz, welche auf einer virtuellen Maschine oder in einer containerisierten Computing-Umgebung betrieben wird [7, Kap. 1.2]. Im ersten Schritt des Pipeline-Prozesses wird die Applikationen zu einem ausführbaren Programm kompiliert (*Artefakt*) (*Build*: s. Abb. 5). Dafür können je nach Programmiersprache verschiedene Build-Tools, wie Maven für Java oder NPM für Javascript verwendet werden [7, Kap. 7.1]. Nach Ablauf der Build-Workflows erfolgt eine automatische Abwicklung des Validierungsprozesses (*Smoke-Tests*). Damit soll sichergestellt werden, dass zu jeder Zeit ein rudimentär getesteter Code bereitsteht und grundlegende Funktionalitäten sowie Schnittstellen erwartungsgemäß ausgeführt werden [5, S. 19]. Der in dem Entwicklungszweig bereitgestellte Code wird dabei überwiegend anhand schnell durchführbarer Tests

überprüft. Der Zweck dieser zügigen Validierungen liegt dabei insbesondere darin, dass Entwickler zeitnahes Feedback auf die Erweiterungen erhalten. So können Fehler und Konflikte so schnell wie möglich entdeckt und behoben werden, was die Entwicklung bei einer reibungslosen Auslieferung der IT-Services unterstützt. Die in der CI-Pipeline abgewickelten Validierungen umfassen i.d.R. *Unit-* sowie *Integration-Tests* [7, Kap. 1.2].

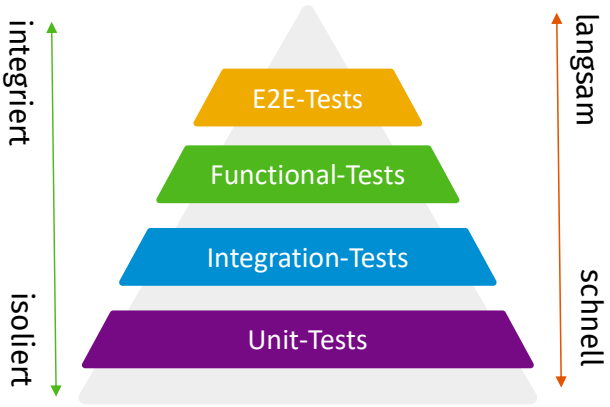

Abbildung 7: Hierarschische Darstellung von Softwaretests.
In Anlehnung an Paspelava [21].

Unit-Tests befinden sich dabei auf unterster Hierarchieebene der Test-Pyramide (s. Abb. 7). Somit besitzen diese eine kurze Ausführungsdauer, werden jedoch ausschließlich in einer isolierten Testumgebung abgewickelt. Mit Unit-Tests wird die funktionale Korrektheit kleinster Einheiten, wie z.B. Methoden einer Klasse, überprüft. Der Zweck der Unit-Tests besteht dabei in einer von externen Einflüssen und Daten unabhängigen Überprüfung der einzelnen Komponenten [6, Kap. 2]. Um bei der Bereitstellung neuer Funktionalitäten ebenfalls das Zusammenspiel verschiedener Komponenten zu überprüfen, werden *Integration-Tests* durchgeführt. Bei diesen Tests können Aspekte, wie der Austausch eines Nachrichtenmodells zweier Web-Services oder das Response-Objekt einer Datenbankabfrage untersucht werden [6, Kap. 2]. Nachdem einzelne Funktionalitäten entwickelt und alle Tests erfolgreich absolviert wurden, werden die validierten Änderungen im Hauptzweig zusammen-

geführt. Mit diesem Prozessschritt beginnt der *Continuous-Delivery-Workflow (CD-Workflow)*. Während CI den Prozess der kontinuierlichen Integration des Quellcodes in das zentrale Repository verwaltet, steuert der CD-Workflow die Automatisierung der Anwendungsbereitstellung. Applikationen sollen somit ohne große Verzögerungen in die Produktivumgebung und somit zum Kunden ausgeliefert werden. Im Sinne des DevOps-Rahmenwerkes wird der CD-Prozess automatisch und unmittelbar nach Ablauf aller CI-Aktivitäten angestoßen. In der Praxis wird hierbei jedoch häufig ein manueller Schritt zwischengeschalten [5, S. 20]. Damit soll sichergestellt werden, dass das Ausrollen der Anwendung erst nach Überprüfung und Genehmigung der Product Owner beginnt. Im ersten Schritt des CD-Prozesses wird das in die Produktivumgebung bereitzustellende Artefakt über die Deployment-Pipeline in eine *Staging-Area* geladen. Bei der Staging-Area handelt es sich dabei um ein System, welches zwischen Entwicklungs- und Produktivumgebung liegt. Die Staging-System-Konfigurationen werden dabei so angelegt, dass diese der Produktionsumgebung möglichst ähnlich sind [7, Kap. 1.3]. Neben den Datenbanken werden hierbei ebenfalls Serverkonfigurationen, wie Firewall- oder Netzwerkeinstellungen von dem Produktivsystem übernommen. Somit soll sichergestellt werden, dass eine neue Anwendungsversion unter produktions-ähnlichen Bedingungen getestet wird. Analog zum CI-Prozess werden innerhalb des CD-Workflows ebenfalls Unit- und Integration-Tests abgewickelt. Diese sind i.d.R. deutlich rechenintensiver und besitzen längere Ausführungszeiten. Somit werden im CD-Prozess essenzielle, jedoch während des Entwicklungsworkflows zu aufwendige Validierungen durchgeführt [5, S. 20]. In der Staging Area werden unterdessen auch in der Test-Pyramide (s. Abb. 7) höher positionierte, also rechenintensivere Tests ausgeführt [6, Kap. 2]. Dazu gehören *Functional-Tests*. Mit diesen werden die in der Planungsphase festgelegten Anforderungen bzw. Funktionen der Anwendung überprüft. So kann z.B. evaluiert werden, ob bei Eingabe einer Benutzer-Passwort-Kennung ein korrekter Autorisierungstoken übergeben wurde. Genau wie bei Integration-Tests wird während Functional-Tests das Zusammenspiel verschiedener Komponenten überprüft. Bei Integration-Tests wird dabei jedoch lediglich die generelle Durchführbarkeit einer Kommuni-

kation verschiedener Komponenten auf Quellcode bzw. Datenbankebene überprüft. Mit Functional-Tests wird darüber hinaus die nach Übermittlung und Prozessierung der verschiedenen Komponenten generierte Ausgabe auf Ebene des Gesamtsystems validiert. Ebenfalls während des CD-Prozesses ausgeführte Validierungen sind *End-to-End-Tests (E2E-Tests)*. Mit diesen soll sichergestellt werden, dass die Anforderungen aller Stakeholder erfüllt werden. Hierbei wird ein vollständiges Anwenderszenario von Anfang bis Ende getestet. Dieses kann im Kontext eines E-Commerce-Webshops etwa das Anmelden mit Benutzername, das Suchen eines Produktes und das Abschließen einer Bestellung umfassen [18]. Nachdem alle Unit-, Integration-, sowie Functional-Tests erfolgreich absolviert wurden, werden i.d.R. verschiedene Codeanalysen angestoßen. Hierbei werden Metriken, wie die prozentuale Testabdeckung oder Schwachstellen verwendeter Code-Patterns überprüft. Nach Durchführung der Codeanalysen wird das überprüfte Artefakt auf die Cloud-Plattform geladen (*Deploy*: s. Abb. 5). Je nach Bereitstellungsstrategie (s. 2.2.3), wird die Anwendung dann unmittelbar oder erst nach weiteren Überprüfungen für den Kunden zugänglich gemacht. Der letzte Schritt des CD-Workflows umfasst die Laufzeitüberwachung der inbetriebgenommenen Anwendung (*Monitoring*: s. Abb. 5). So soll eine ordnungsgemäße Ausführung der Anwendung in der Produktionsumgebung sichergestellt werden. Wichtige Überwachungselemente sind dabei *Infrastruktur-*, *Plattform-* sowie *Anwendungs-Monitoring*. Beim Infrastruktur-Monitoring werden Metriken wie CPU-, Speicher- und Netzwerklast der Server bzw. Datenbanken untersucht. Das Plattform-Monitoring setzt dabei eine Ebene höher an und validiert, dass die Plattform, auf welcher die Anwendungen ausgeführt werden, stabil und fähig ist. Dabei werden Infrastrukturkomponenten, wie Datenbanken, virtuelle Netze bzw. Middleware analysiert. Das Anwendungs-Monitoring umfasst hingegen die Überwachung der Funktionalitäten und der Applikation selbst. Hierbei werden Informationen wie Anfragen pro Sekunden, die Anzahl der Benutzer oder die in Log-Dateien gesammelte Fehlercodes analysiert [5, S. 21].

2.2.3 Strategien zur Bereitstellung von Neuentwicklungen

Nachdem das Artefakt auf einer virtuellen Maschine bzw. containerisierten Cloud-Instanz installiert und gestartet wurde, erfolgt die Inbetriebnahme der neuen Anwendungsversion je nach Bereitstellungsstrategie unmittelbar oder erst nach weiteren Validierung. Anhand der Bereitstellungsstrategie wird festgelegt, mit welcher Methode und zu welchem Zeitpunkt Nutzeranfragen von der aktuellen auf die neue Anwendungsinstanz umgeleitet werden.

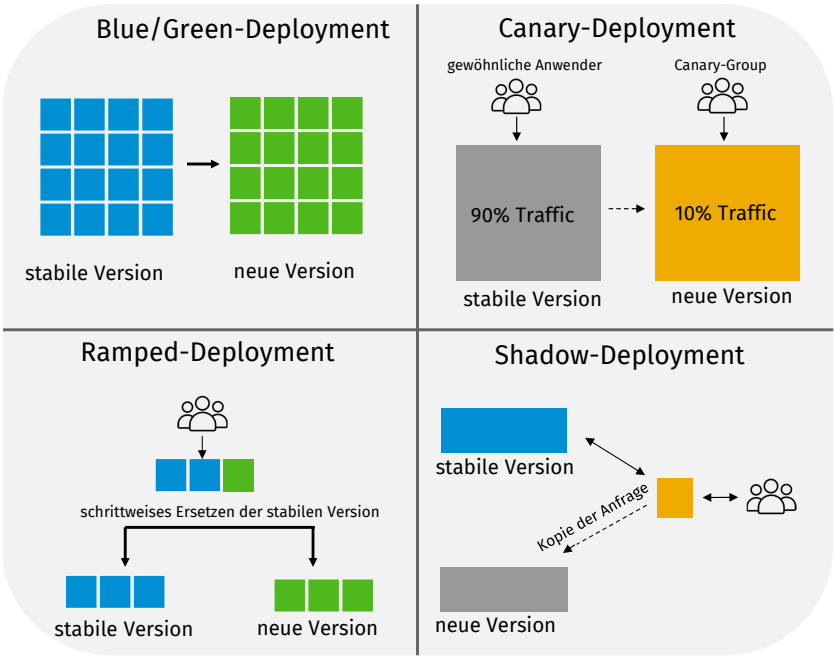

Abbildung 8: Strategien zur Bereitstellung von Software.
In Anlehnung an Ugochi [24].

Eine häufig verwendete Deployment-Strategie ist dabei das *Blue/Green-Deployment*. Hierbei wird neben der stabilen aktuellen Anwendung (*Blaue Version*) ebenfalls eine Instanz der neuen Anwendung (*Grüne Version*) betrieben. Nutzeranfragen werden dabei von dem Lastenverteilungsservice (*Load-Balancer*) erst nach Validierung aller Tests umgeschalten. Dazu gehören neben den in der CI/CD-Pipeline definierten Tests ebenfalls Überprüfungen der Qualitätssicherung. Diese umfassen manuelle Tests, in welchen Funktionen, Benutzeroberfläche sowie die Anwenderfreundlichkeit überprüft werden [24]. Im Gegensatz zum Blue/Green-Deployment, bei wel-

chem eine neue Version simultan für die gesamte Nutzerbasis zur Verfügung gestellt wird, gewährleistet das *Canary-Deployment* eine restriktivere Nutzlastumleitung. Hierfür wird die neue Anwendungsversion vorerst einer überschaubaren Nutzeranzahl (*Canary-Gruppe*) bereitgestellt. Dabei sollte die zusammengestellte Canary-Gruppe die Gesamtnutzerbasis möglichst gut repräsentieren. Anhand des Canary-Traffics soll der fehlerfreie Betrieb neuer Anwendungen überprüft und ggf. Anpassungen vorgenommen werden, bevor diese der gesamten Nutzerbasis zur Verfügung gestellt werden [24]. Für Anwendungen auf einer kritischen IT-Infrastruktur wird i.d.R. die *Ramped-Deployment-Strategie* verwendet. Diese ermöglicht eine präzise Kontrolle horizontal skalierter Services. Bei einer horizontalen Skalierungen erfolgt die Replizierung eine identischen Service-Instanzen, wodurch die Ausfallsicherheit einer Anwendung optimiert werden kann. Die neue Softwareversion wird während des Ramped-Deployment-Prozesses schrittweise auf die horizontalen Instanzen ausgerollt. Dabei werden die ersten aktualisierten Instanzen lediglich für bestimmte Anwender, eine sog. *Ramped-Gruppe*, bereitgestellt. Dabei soll das von dieser Anwendergruppe zur Verfügung gestellte Feedback während zukünftiger Planungsprozesse berücksichtigt werden [24]. Eine aufwendigere, jedoch risikoärmere Bereitstellungsstrategie stellt das *Shadow-Deployment* dar. Dabei wird neben der Instanz der aktuellen Version ebenfalls ein sog. *Shadow-Model* auf der Infrastruktur betrieben. Das Shadow-Model verwaltet die neue Version der Anwendung, kann jedoch nicht unmittelbar von den Nutzern aufgerufen werden. Diese Instanz stellt ein hinter der stabilen Version gelagertes Schattenmodell dar. Benutzeranfragen werden von dem Load-Balancer stets auf die aktuelle Version der Instanz weitergeleitet, verarbeitet und beantwortet. Gleichzeitig wird eine Kopie dieser Anfrage an das Shadow-Model weitergeleitet und von diesem prozessiert. Die Shadow-Modell-Verarbeitung des in der Produktionsumgebung abgewickelten Netzwerkverkehrs ermöglicht den Entwicklern somit eine anwendungsbezogene Überprüfung entwickelter Features [24].

3 Methodische Vorgehensweise

3.1 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines

3.2 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses

3.3 Semistrukturierte Leitfadeninterviews

4 Anwendung der Methodik auf die theoretischen Grundlagen

4.1 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines

4.2 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses

4.3 Entwicklung einer ganzheitlichen Bereitstellungsstrategie

5 Schlussbetrachtung

5.1 Fazit und kritische Reflexion

5.2 Ausblick

Literatur

Print-Quellen

[1] Matthias Biehl. *API architecture. The big picture for building APIs*. en. API-university series. API-University Press, 2015. ISBN: 9781508676645.

[2] Ralf Bruns und Jürgen Dunkel. *Event-Driven Architecture. Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. de. Xpert.press. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 9783642024399. DOI: 10.1007/978-3-642-02439-9.

[3] Rong N. Chang u. a. „Realizing A Composable Enterprise Microservices Fabric with AI-Accelerated Material Discovery API Services“. In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 2020 IEEE 13th International Conference on Cloud Computing (CLOUD) (Beijing, China). IEEE, 10/19/2020 - 10/23/2020, S. 313–320. ISBN: 978-1-7281-8780-8. DOI: 10.1109/CLOUD49709.2020.00051.

[4] Joachim Goll und Daniel Hommel. *Mit Scrum zum gewünschten System*. ger. Wiesbaden: Springer Vieweg, 2015. 185 S. ISBN: 9783658107208. DOI: 10.1007/978-3-658-10721-5.

[5] Jürgen Halstenberg. *DevOps. Ein Überblick*. ger. Unter Mitarb. von Bernd Pfitzinger und Thomas Jestädt. Essentials Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020. 159 S. ISBN: 9783658314057. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6380828.

[6] Brian Hambling und Brian, Hrsg. *Software testing. An ISTQB-BCS certified tester foundation guide*. eng. Unter Mitarb. von Brian Hambling. 3rd ed. Hambling, Brian (MitwirkendeR) Brian, (author.) Hambling, Brian, (editor.) London, England: BCS Learning & Development Limited, 2015. 11 S. ISBN: 9781780173016. URL: https://learning.oreilly.com/library/view/-/9781780172996/?ar.

[7] Mohamed Labouardy. *Pipeline as code. Continuous delivery with Jenkins, Kubernetes, and Terraform*. eng. Shelter Island, New York: Manning Publications Co, 2021. 1333 S. ISBN: 9781638350378. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6785307.

[8] Jon Loeliger und Matthew McCullough. *Version control with git. Powerful tools and techniques for collaborative software development*. en. 2nd ed. Sebastopol, CA.: O'Reilly, 2012. 434 S. ISBN: 9781449345051.

[9] Dieter Masak. *Digitale Ökosysteme. Serviceorientierung bei dynamisch vernetzten Unternehmen*. Xpert.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-79129-4. DOI: 10.1007/978-3-540-79130-0.

[10] Stefan Reinheimer. *Cloud Computing. Die Infrastruktur der Digitalisierung*. Edition HMD. Wiesbaden, Germany und Ann Arbor: Springer Vieweg und ProQuest EbookCentral, 2018. ISBN: 978-3-658-20966-7. DOI: 10.1007/978-3-658-20967-4.

[11] Sensedia, Hrsg. *The Future is Composable: How APIs, Microservices and Events are reshaping the next-gen Enterprises*. 2020. URL: https://f.hubspotusercontent30.net/hubfs/4209582/%5BInternational%5D%20Boardroom%20Nordics/%28EN%29%20Composable%20Enterprises%20-%20Sensedia.pdf.

[12] Joseph T. Vesey. „Time-to-market: Put speed in product development". In: *Industrial Marketing Management* 21.2 (1992). PII: 001985019290010Q, S. 151–158. ISSN: 0019-8501. DOI: 10.1016/0019-8501(92)90010-Q. URL: https://www.sciencedirect.com/science/article/pii/001985019290010q.

[13] Wolfgang Vieweg. „Agiles (Projekt-)Management". de. In: *Management in Komplexität und Unsicherheit*. Springer, Wiesbaden, 2015, S. 41–42. DOI: 10.1007/978-3-658-08250-5_11. URL: https://link.springer.com/chapter/10.1007/978-3-658-08250-5_11.

[14] Alberto Vivenzio, Hrsg. *Testmanagement Bei SAP-Projekten. Erfolgreich Planen * Steuern * Reporten Bei der Einführung Von SAP-Banking*. ger. Unter

Mitarb. von Domenico Vivenzio. 1st ed. Vivenzio, Alberto (VerfasserIn) Vivenzio, Domenico (MitwirkendeR). Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2013. 1175 S. ISBN: 978-3-8348-1623-8. DOI: 10.1007/978-3-8348-2142-3.

[15] Fiorella Zampetti u. a. „CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (Luxembourg). IEEE, 9/27/2021 - 10/1/2021, S. 471–482. ISBN: 978-1-6654-2882-8. DOI: 10.1109/ICSME52107.2021.00048.

Online-Quellen

[16] John Adam. *Was ist agile Softwareentwicklung?* Hrsg. von K&C. 2021. URL: https://kruschecompany.com/de/agile-softwareentwicklung/ (besucht am 05. 03. 2023).

[17] Shweta Bhanda und Abby Taylor. *The Evolution from Agile to DevOps to Continuous Delivery — Qentelli*. Hrsg. von Qentelli. 2023-03-05. URL: https://www.qentelli.com/thought-leadership/insights/evolution-agile-devops-continuous-delivery (besucht am 05. 03. 2023).

[18] Shreya Bose. *What is End To End Testing?* BrowserStack. 2023-02-20. URL: https://www.browserstack.com/guide/end-to-end-testing (besucht am 08. 03. 2023).

[19] Johannes Klingberg. *Composable Enterprise: Warum das Unternehmen der Zukunft modular aufgebaut ist*. Hrsg. von Magnolia. 2021. URL: https://www.magnolia-cms.com/de_DE/blog/composable-enterprise.html (besucht am 13. 03. 2023).

[20] McKinsey, Hrsg. *The business value of design*. 2019. URL: https://www.mckinsey.com/capabilities/mckinsey-design/our-insights/the-business-value-of-design (besucht am 05. 03. 2023).

[21] Darya Paspelava. *What is Unit Testing in Software. Why Unit Testing is Important.* Hrsg. von Exposit. 2021. URL: https://www.exposit.com/blog/what-unit-testing-software-testing-and-why-it-important/ (besucht am 08. 03. 2023).

[22] Jörg Schönenstein. *Composable-Business und -Commerce durch MACH-Architektur.* Hrsg. von communicode AG. 2023. URL: https://www.communicode.de/blog/work/composable-business-und-commerce-durch-mach-technologie (besucht am 13. 03. 2023).

[23] Synopsys, Hrsg. *What Is CI/CD and How Does It Work?* 2023-02-01. URL: https://www.synopsys.com/glossary/what-is-cicd.html (besucht am 08. 03. 2023).

[24] Ukpai Ugochi. *Deployment Strategies: 6 Explained in Depth.* Hrsg. von Plutora. 2022. URL: https://www.plutora.com/blog/deployment-strategies-6-explained-in-depth (besucht am 08. 03. 2023).

Anhang