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

Selbstständigkeitserklärung **II**

Abkürzungsverzeichnis **V**

Abbildungsverzeichnis **VI**

Tabellenverzeichnis **VII**

1 Einleitung **1**

1.1 Motivation und Problemstellung 1

1.2 Zielsetzung und Abgrenzung 1

1.3 Aufbau der Arbeit 1

2 Grundlagen und Begriffserklärungen **2**

2.1 Die Composable-Enterprise-Architektur 2

2.1.1 Begriffserklärung und Abgrenzung 2

2.1.2 Technologische Konzepte des Composable-Enterprises 4

2.2 Integration und Bereitstellung von Software 6

2.2.1 Agile und DevOps als moderne Anwendungsentwicklungskonzepte 6

2.2.2 Pipelines zur Integrations- und Bereitstellungsautomatisierung 10

2.2.3 Strategien zur Bereitstellung von Neuentwicklungen 16

3 Methodische Vorgehensweise **19**

3.1 Semistrukturierte Leitfadeninterviews 19

3.2 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines 21

3.3 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses 21

4 Anwendung der Methodik auf die theoretischen Grundlagen **25**

4.1 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines 25
4.2 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses 25
4.2.1 Festlegung der AHP-Entscheidungsalternativen 25
4.3 Entwicklung einer ganzheitlichen Bereitstellungsstrategie 34

5 Schlussbetrachtung **35**
5.1 Fazit und kritische Reflexion 35
5.2 Ausblick 35

Anhang **XIII**

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
SAP CI/CD	SAP Continuous Integration and Delivery
MTA	Multi-Target Application
SAP CTM	SAP Cloud Transport Management
SAP BAS	SAP Business Application Studio
JaaS	Jenkins-as-a-Service

Abbildungsverzeichnis

1 Entstehung einer Composable-Enterprise-Architektur 2
2 Technische Realisierung der Composable-Enterprise-Architektur . . . 4
3 Exemplarische Abfolge eines agilen Entwicklungszykluses 7
4 Zeitliche Darstellung der Herbeiführung von Kundennutzen bei der Entwicklung von IT-Services 9
5 Aktivitäten im CI/CD-Prozess 10
6 Versionskontrollsysteme zur Verwaltung von Quellcode 12
7 Hierarschische Darstellung von Softwaretests 13
8 Strategien zur Bereitstellung von Software 17
9 Exemplarische Darstellung der Paarvergleichsmatrix im AHP 22
10 Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP 23
11 AHP-Entscheidungsstruktur zur Bewertung von CI/CD-Pipelines . . 26
12 Bereitstellung von MTA-Applikationen in ein Artefakt-Repository . . 28
13 SAP Cloud Transportmanagement 29

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“ [22]. Für Analystenhäuser 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)**.

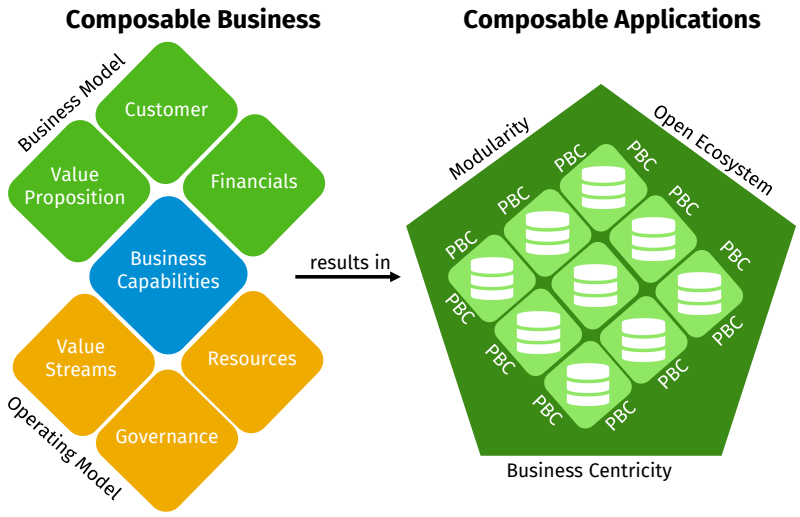

Abbildung 1: Entstehung einer Composable-Enterprise-Architektur. In Anlehnung an Schönstein [26].

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 [23].“

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* [23]. 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" [23]. Ergibt sich eine Änderung in den Geschäftsanforderungen, ermöglicht diese Architektur ein flexibles und isoliertes Austauschen, Verändern sowie Weiterentwickeln einzelner PBCs. Um eine auf den Geschäftszweck maßgeschneiderte IT-Lösung zu implementieren, werden diese spezialisierten Komponenten kombiniert und miteinander 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 benötigte Komponente, wie z.B. PBCs 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 benötigt 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 unmittelbar bereitgestellt werden [11, S. 58]. Dazu gehören etwa Tools zur Automatisierung von Prozessen oder Kollaborationsdienste zur Unterstützung der Zusammenarbeit innerhalb von Teams. Neben diesen administrativen Werkzeugen werden auf Marktplätzen ebenfalls offene Technologiestandards, wie Datenbanken oder Sicherheitstools angeboten. Diese werden von CEs als Werkzeuge 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 [14, S. 7]. Um eine anwenderzen-

trierte 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 kann dies zu Ineffizienzen innerhalb der Arbeitsprozesse führen. Deshalb ist essenziell, dass Mitarbeiter Systeme intuitiv nutzen und ggf. weiterentwickeln und anpassen können, ohne dabei von der IT-Abteilungen abhängig zu sein [23].

2.1.2 Technologische Konzepte des Composable-Enterprises

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

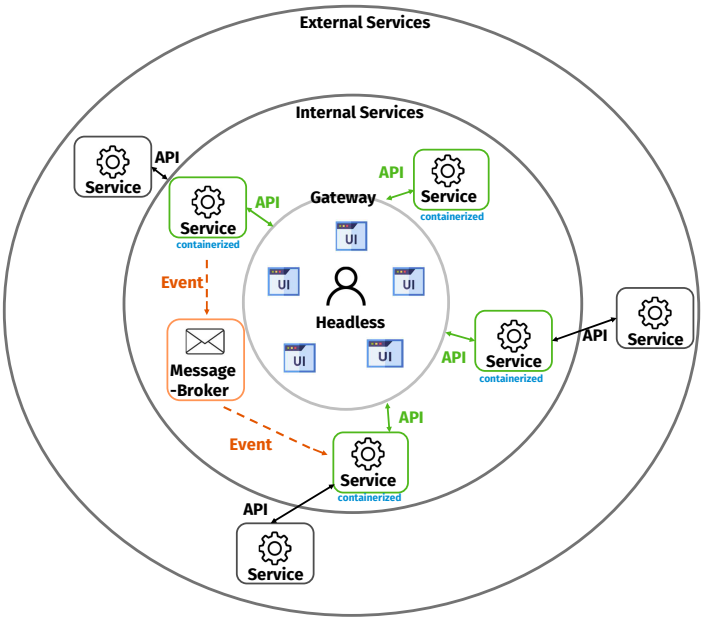

Abbildung 2: Technische Realisierung der Composable-Enterprise-Architektur. Eigene Darstellung.

Der zentrale Einstiegspunkt des Nutzers (Client) in einer CE-Anwendung ist das User-Interface. 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 [23]. Die Applikationslogik des Backends wird dabei in kleine isolierte *Microservices* gekapselt. Um Nachrichten zwischen Frontend und den Services zu übermitteln, wird i.d.R. ein Gateway zwischengeschaltet [11, S. 41]. Mit diesem wird eine zentrale und standardisierte Schnittstelle bereitgestellt, welche verschiedene Authentifizierungs-, Autorisierungs- sowie Routing-Mechanismen implementiert. Die einzelnen Services werden auf losen Instanzen betrieben, welche jedoch im Zusammenspiel eine große Anwendung darstellen. Dabei ist möglich, 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 die 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 Dienste 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 [12, 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)* [12] [12, 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 **BTP!** (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 [12, 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 Vor-

habens den Spielraum für Anpassungsmöglichkeiten [17, S. 5]. Dies resultiert nicht nur einem Anstieg der Kosten, sondern führt ebenfalls dazu, dass IT-Projekte länger als geplant ausfallen [16, 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) [5, 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.

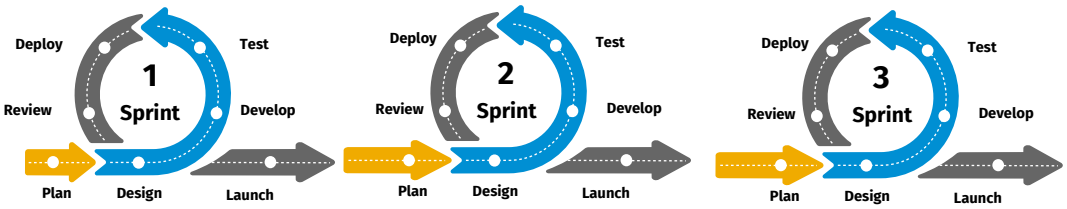

Abbildung 3: Exemplarische Abfolge eines agilen Entwicklungszykluses. In Anlehnung an K&C [19].

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 [19, 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 [20]. 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 [6, 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 [24]. Abhilfe schaffen kann das in der Literatur als **Development & Operations (DevOps)** bekannte Aufbrechen organisatorischer Silos zwischen Entwicklung und dem IT-Betrieb [6, 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)* [6, 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 geeignetsten sind, Dispositionen zu verabschieden [6, 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

[6, S. 7].

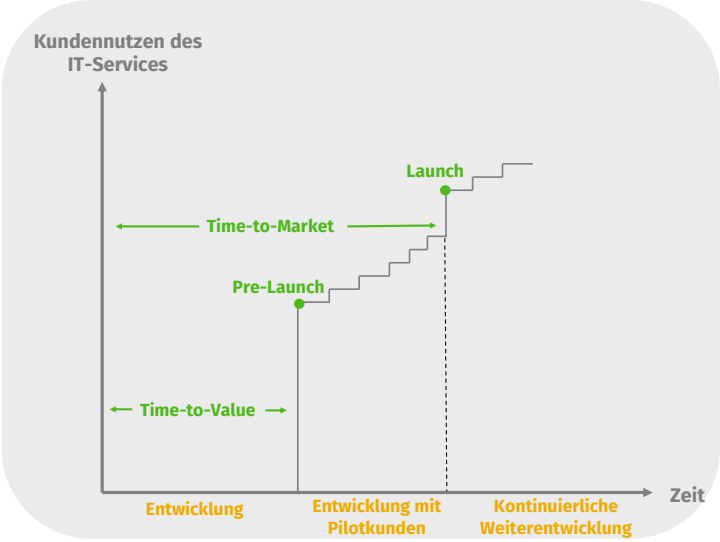

Abbildung 4: Zeitliche Darstellung der Herbeiführung von Kundennutzen bei der Entwicklung von IT-Services [6, S. 9].

Der Time-to-Market beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [15, 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 [6, 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 [15]. Aus

diesen Erfahrungen sollen Best-Practises entwickelt werden, welche innerhalb von Teams und organisationsübergreifend weitergegeben werden (*Teilen*) [6, 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 [18, S. 471].

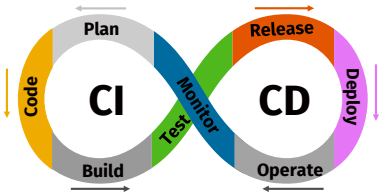

Abbildung 5: Aktivitäten im CI/CD-Prozess. In Anlehnung an Synopsys [27].

Alle in diesem Prozess anfallenden Aktivitäten werden dabei im CI/CD-Zyklus der Abb. 5 dargestellt. 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 [18, 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 [6, 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 [10, 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 [10, 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.

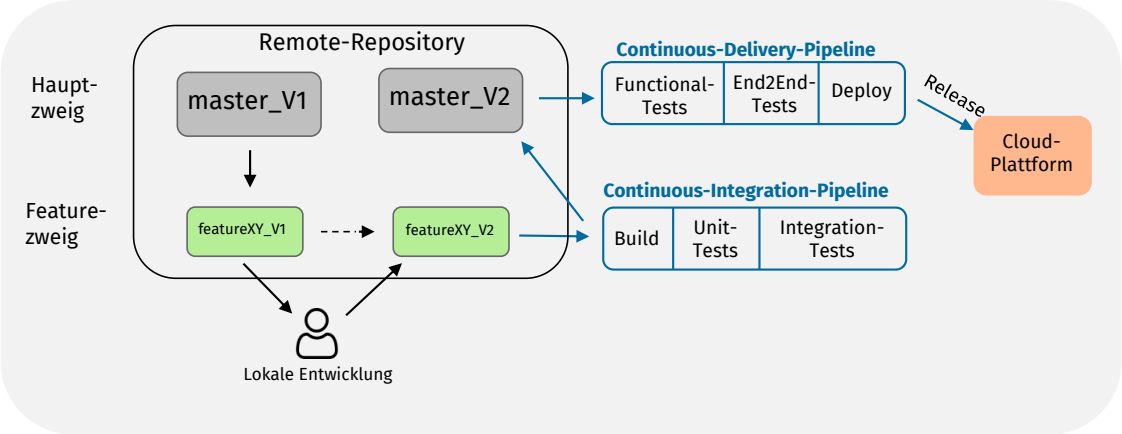

Abbildung 6: Versionskontrollsysteme zur Verwaltung von Quellcode. Eigene Darstellung.

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 [9, 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 NPM für JavaScript oder Make für Multi-Target Application (MTA) verwendet werden [9, 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 [6, 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* [9, Kap. 1.2].

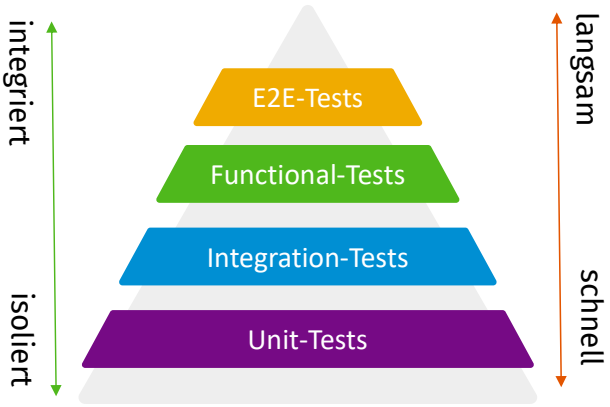

Abbildung 7: Hierarschische Darstellung von Softwaretests.
In Anlehnung an Paspelava [25].

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 [7, 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 [7, Kap. 2]. Im CI-Prozess werden i.d.R. auch einfache Code-Analysen durchgeführt. Diese sollen dem Entwickler eine schnelle Rückmeldung bezüglich Verletzung von Qualitätsstandards, potenziellen Schwachstellen sowie Leistungsproblemen liefern. Nachdem einzelne Funktionalitäten entwickelt und alle Tests erfolgreich absolviert wurden, werden die validierten Änderungen im Hauptzweig zusammengeführt. 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 [6, 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 [9, 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 [6, S. 20]. In der Staging Area werden unterdessen auch in der Test-Pyramide (s. Abb. 7) höher positionierte, also rechenintensivere Tests ausgeführt [7, 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 Kommunikation 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 [21]. Für kritische Systeme werden während des Delivery-Prozesses ebenfalls *Regression-Tests* vorgenommen. Diese umfassen ein erneutes Testen bereits vorhandener Software-Komponenten. Regression-Tests können dabei in Form von Unit-, Integration- sowie Functional-Tests ausgeführt werden. Nachdem alle 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 funktionsfähig ist. Dabei werden Infrastrukturkomponenten, wie Datenbanken, virtuelle Netze bzw. Middlewares analysiert. Das Anwendungs-Monitoring umfasst die Überwachung der Funktionalitäten und der Applikation selbst. Hierbei werden Informationen wie Anfragen pro Sekunden, die Anzahl der Benutzer oder die in Log-Dateien gesammelten Fehlercodes analysiert [6, S. 21].

Zur Automatisierung der CI/CD-Prozesse werden bei der SAP i.d.R. drei verschiedene Pipeline-Tools verwendet. Eine unmittelbare von der SAP bereitgestellte Lösung ist das *SAP Continuous Integration and Delivery (SAP CI/CD)*. Das SAP CI/CD ist

eine auf der SAP BTP betriebene SaaS-Lösung, mit welcher vordefinierte Pipeline-Templates konfiguriert und ausgeführt werden können. Dieses Tool ist insbesondere mit SAP-Standardtechnologien, wie den Programmierframeworks SAP UI5 (Frontend) und SAP CAP Node (Backend) sowie der Laufzeitumgebung Cloud-Foundry kompatibel. Eine weitere bei der SAP verwendete CI/CD-Alternative ist das Open-Source-Tool *Jenkins*. Im Gegensatz zum templatebasierten SAP CI/CD, muss der Bereitstellungsworkflow bei Jenkins mit der Programmiersprache Groover implementiert werden. Weiterhin wird Jenkins nicht unmittelbar auf der SAP BTP betrieben, sondern muss auf einem eigenen Server (On-Premise) oder von einem externen Cloud-Anbieter verwaltet werden. Um die Bereitstellung von SAP-spezifischen Technologien zu optimieren, wurde von der SAP die Programmbibliothek *Project Piper* entworfen. Diese Bibliothek umfasst hoch konfigurierbare Implementationsschritte, Szenarien und Dienstprogramme welche für das Ausrollen von SAP-Anwendungssoftware essenziell sind. Ein externes ebenfalls innerhalb der SAP verwendetes CI/CD-Werkzeug ist *Azure Pipelines*. Azure Pipelines ist ein von Microsoft entwickeltes Tools, welches umfassende Integrationsmöglichkeiten zu anderen Microsoft-Diensten wie die Azure-Cloud-Platform oder Microsoft Visual Studio Code bietet. Zur Implementierung des CI/CD-Workflows SAP-spezifischer Technologien wird für Azure Pipelines ebenfalls die Programmbibliothek Project-Piper verwendet.

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.

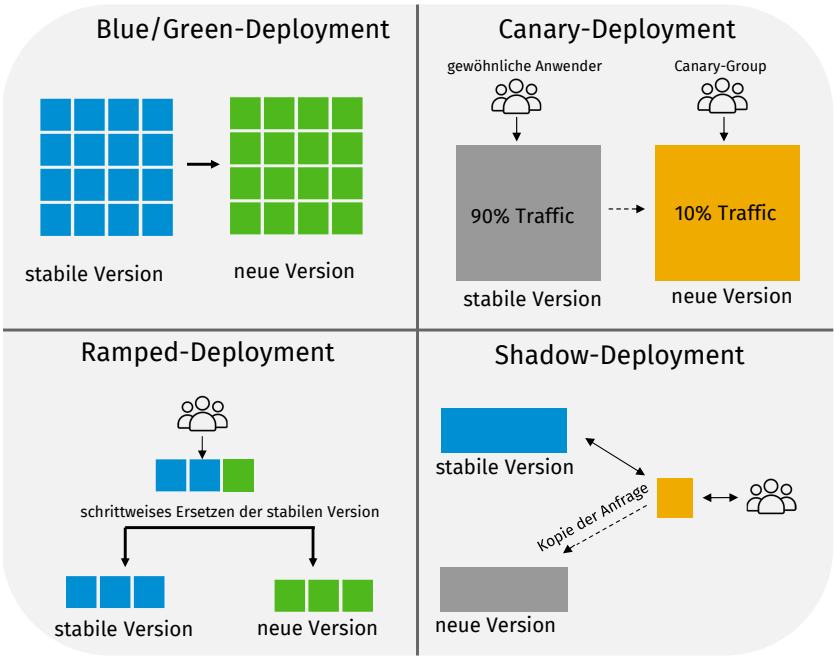

Abbildung 8: Strategien zur Bereitstellung von Software.
In Anlehnung an Ugochi [28].

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 [28]. Im Gegensatz zum Blue/Green-Deployment, bei welchem 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 [28]. 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 [28]. 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 [28].

3 Methodische Vorgehensweise

Der Forschungsbereich dieser wissenschaftlichen Abhandlung umfasst die Themengebiete CI/CD sowie CEA. Da in Kombination dieser beiden Forschungsbereiche sowie in der praktischen Umsetzung dieser Konzepte mit SAP-spezifischen Technologien in der Literatur kein Datenmaterial vorhanden ist, werden im Rahmen dieser Arbeit Experteninterviews durchgeführt. Die in diesen Gesprächen erhobenen Daten sollen dabei als Entscheidungsgrundlage zur Durchführung des AHP-Verfahrens verwendet werden.

3.1 Semistrukturierte Leitfadeninterviews

Das Experteninterview stellt eine häufig angewandte Analysemethode dar, welche vorrangig bei qualitativen Untersuchungen verwendet wird. Die Meinungen, Erfahrungen und Perspektiven der Experten werden dazu verwendet, relevante Aspekte zu einem Thema zu identifizieren oder eine Forschungshypothese zu formulieren. Diese wissenschaftliche Methode wird dabei insbesondere für aktuelle, stets unerforschte Themen sowie Fragestellungen mit geringem Literaturaufkommen verwendet [4, 363 ff.]. Als Experten werden in diesem Zusammenhang Interviewpartner bezeichnet, welche aufgrund ihres im Rahmen beruflicher Tätigkeiten erworbenen Wissens umfassende Kenntnisse in einem spezifischen Fachgebiet besitzen. In der Literatur werden verschiedene Arten von Experteninterviews definiert. Dazu gehören strukturierte, semistrukturierte sowie unstrukturierte Interviews [4, 363 ff.]. Strukturierte Expertengespräche zeichnen sich dabei insbesondere durch die Vorabfestlegung der im Interview gestellten Fragen aus. Hierbei wird bezweckt, dass allen Teilnehmenden dieselben Fragen in standardisierter Reihenfolge vorgelegt werden. Im anderen Extrem der unstrukturierten Interviews erfolgt lediglich eine Definition des Forschungsbereichs, jedoch werden vorab keine expliziten Fragen festgelegt. Den konkreten Verlauf des Gespräches bestimmt dabei die dynamische Entwicklung des Antwort-Nachfrage-Verhaltens der Interviewteilnehmer. Aufgrund des eng abgegrenzten Forschungsbereichs, besteht bei der Durchführung von unstrukturierten

Interviews in dieser Arbeit das Risiko, dass die Gesprächsinhalte vom eigentlichen Untersuchungsgegenstand abweichen. Auch die Abwicklung von strukturierten Interviews ist für diese Arbeit nicht geeignet [8, 244 ff.]. Das liegt insbesondere an dem starren Erhebungsdesign der strukturierten Interviews. Angesichts der in dieser Arbeit angestrebten Erschließung unerforschter Themengebiete bietet eine Vorabfestlegung der Fragen nicht genügend Flexibilität, um auf neu aufkommende Aspekte innerhalb des Gesprächs angemessen zu reagieren. Deshalb werden im Rahmen dieser Arbeit semistrukturierte Interviews durchgeführt. Diese Interviewform realisiert eine Leitfadenstruktur, welche eine vordefinierte Fragegestaltung vorgibt, jedoch im Verlauf des Gesprächs gleichzeitig ein hohes Maß an Flexibilität bietet. Ergeben sich während eines Expertengesprächs neue Aspekte, können diese mit unmittelbaren Ad-hoc-Fragen aufgegriffen werden. Um die Interviews auszuwerten, muss eine Transkription der Expertengespräche erfolgen [8, 244 ff.]. Dabei gibt es neben der lautsprachlichen bzw. vereinfachenden ebenfalls eine zusammenfassende Transkription. Während in der lautsprachlichen Transkription Gespräche in vollständiger Form erfasst werden, kennzeichnet sich eine vereinfachende Transkription durch das Korrigieren von Dialekten, Satzbrüchen oder Wortdopplungen. Demgegenüber werden bei einer zusammenfassenden Transkription nur essenzielle Gesprächsinhalte festgehalten. Da zur Beantwortung der Forschungsfrage dieser Arbeit nicht der exakte Wortlaut, sondern vielmehr die inhaltliche Ausgestaltung der Expertengespräche von Bedeutung ist, wird eine zusammenfassende Transkription durchgeführt. Zur Auswertung der Transkription wird dabei i.d.R. eine deduktive bzw. induktive Methode verwendet. Bei einer deduktiven Auswertung werden Aussagen der Experten vordefinierten Kategorien zugeordnet [8, 244 ff.]. Dieses Evaluationsverfahren bietet insbesondere zur Validierung einer vorgegebenen Forschungshypothese einen erheblichen Mehrwert. Da im Rahmen dieser Arbeit stattdessen die Beantwortung einer offenen Forschungsfrage vorgesehen ist, wird eine induktive Kodierung der Interviews vorgenommen. Statt Themengruppen im Voraus festzulegen, werden bei der induktiven Kodierung Kategorien dynamisch aus dem Interviewmaterial abgeleitet. Eine implizite Auswertung der induktiven Kodierung wird im AHP-Verfahren (s.

Kap. 4.2) angewendet. So werden die während der Evaluation getroffenen Entscheidungen anhand der Expertenaussagen referenziert.

3.2 Prototypische Implementierung der Integrations- und Bereitstellungs-Pipelines

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

AHP ist ein von dem Mathematiker Thomas Saaty konzipiertes Entscheidungs-Framework. Dieses Rahmenwerk eignet sich insbesondere für komplexe betriebswirtschaftliche und technische Entscheidungsprobleme. Bei AHP wird eine Bewertung der Entscheidungsalternativen anhand verschiedener Kriterien vorgenommen. Da das zugrundeliegende Rahmenwerk auf der Prämisse einer divergierenden Wichtigkeit der unterschiedlichen Entscheidungskriterien basiert, muss für die festgelegten Kriterien eine Gewichtung vorgenommen werden. [13, S. 86]. Das AHP-Verfahren besteht dabei aus mehreren Schritten. Zu Beginn des AHP-Verfahrens ist eine exakte Definition der zu lösenden Problemstellung notwendig. Im nächsten Schritt werden verschiedene Entscheidungsalternativen sowie Bewertungskriterien festgelegt. Die Entscheidungsstruktur kann dabei durch einen hierarchischen Aufbau beliebig gestaltet. Auf oberster Ebene des AHP-Baums befindet sich das Entscheidungsproblem (s. Abb. 10). Diese Stufe wird dabei durch mehrere, die Verzweigung der Bewertungskriterien umfassenden Hierarchieebenen gefolgt. So besteht die Möglichkeit, ein Kriterium in mehrere Subkriterien zu unterteilen. Auf unterster Ebene befinden sich schließlich die im Hinblick auf die festgelegten Evaluationskriterien zu bewertenden Entscheidungsalternativen. Um eine auf die Präferenzen der Stakeholder abgestimmte Bewertung zu ermöglichen, müssen die zuvor festgelegten Entscheidungskriterien gewichtet werden. Besteht der AHP-Baum aus mehreren Stufen, erfolgt für jede Ebene zunächst eine isolierte Gewichtung. Zur Bestimmung der relativen Wichtigkeit wird für das AHP-Verfahren nach Saaty ein paarweiser Vergleich vorgeschlagen.

Hierbei wird die Wichtigkeit eines Kriteriums gegenüber eines anderen ermittelt. Die in der Literatur aufgeführte Gegenüberstellung erfolgt dabei auf einer Skala von eins bis neun [13, S. 86]. Um den Prozess der Entscheidungsfindung für die Probanden intuitiver zu gestalten, ist im Rahmen dieser Arbeit eine Gewichtung von null bis zwei vorhergesehen. Während der Wert zwei impliziert, dass ein Entscheidungskriterium wesentlich wichtiger ist, wird mit dem Index eins eine gleiche Gewichtung ausgedrückt. Eine relative Wichtigkeit mit dem Wert null bedeutet in diesem Zusammenhang, dass ein Kriterium weniger wichtig als die Vergleichskategorie ist.

	Kriterium K1	Kriterium K2	Kriterium K3	lok. Gew.
Kriterium K1	1	2	0	0,333
Kriterium K2	0	1	0	0,111
Kriterium K3	2	2	1	0,556

Abbildung 9: Exemplarische Darstellung der Paarvergleichsmatrix im AHP. Eigene Darstellung.

So wird dem in Abb. 10 exemplarisch dargestellten Kriterium K1 eine wesentlichere Bedeutung als K2 zugeschrieben. Das korrespondieren Äquivalent auf der rechten Matrixhälfte besitzt entsprechend eine Gewichtung von null (*weniger wichtig*). Um der in der Paarvergleichsmatrix getroffenen Gegenüberstellung eine höhere Aussagekraft zu verleihen, müssen die relativen Gewichtungen in prozentuale Werte transformiert werden. Im ersten Schritt werden dabei alle Paarvergleichswerte eines Entscheidungskriteriums aufsummiert:

$$V_{k1} = X_{11} + X_{12} + X_{13}$$

Anschließend muss diese Summe standardisiert werden. Hierfür wird die Gewichtungssumme (V_{ki}) eines Kriteriums durch die Gesamtanzahl der in einer Paarvergleichsmatrix (W) vergebenen Punkte dividiert. Diese ergibt sich durch eine Betrachtung der in einer Matrix durchgeführten Paarvergleiche. Beinhaltet eine Matrix

drei Kriterien, werden insgesamt 6 Vergleiche durchgeführt. Da für jeden Paarvergleich stets 2 Punkte vergeben werden, entspricht die Gesamtanzahl der Gewichtungspunkte (W) zwölf ($12 = 2 \cdot 6$). Anschließend kann die prozentuale Gewichtung eines Entscheidungskriteriums wie folgt berechnet werden:

$$W_{k1} = \frac{V_{k1}}{W}$$

Diese lokale Gewichtung wird für jede Hierarchieebene durchgeführt. Um die globale Gewichtung zu ermitteln, wird die Gewichtung jedes Subkriteriums mit den Gewichtungen der übergeordneten Kriterien multipliziert (s. Abb. 10). In der Literatur wird vorgesehen, dass auf unterster Hierarchieebene ebenfalls eine Gewichtung der Entscheidungskriterien vorgenommen wird [13, S. 86]. Da dies insbesondere bei auf subjektiven Präferenzen basierenden Problemstellungen eine wichtige Rolle spielt, wird hierbei von dem Leitfaden nach Saaty abgewichen. Stattdessen wird für jedes Kriterium auf unterster Ebene eine feste Metrik definiert, anhand welcher die Alternativen bewertet werden. Die in der Metrik festgelegte Punktevergabe kann dabei sowohl anhand qualitativer als auch quantitativer Aspekte durchgeführt werden. Letztlich werden die für jedes Kriterium erzielten Punkte mit den globalen Gewichtungsfaktoren multipliziert und alle Teilbewertungen zu einer gewichteten Gesamtbenotung zusammengezogen. Die Entscheidungsalternative mit der höchsten Gesamtbewertung gilt dabei als optimale Alternative.

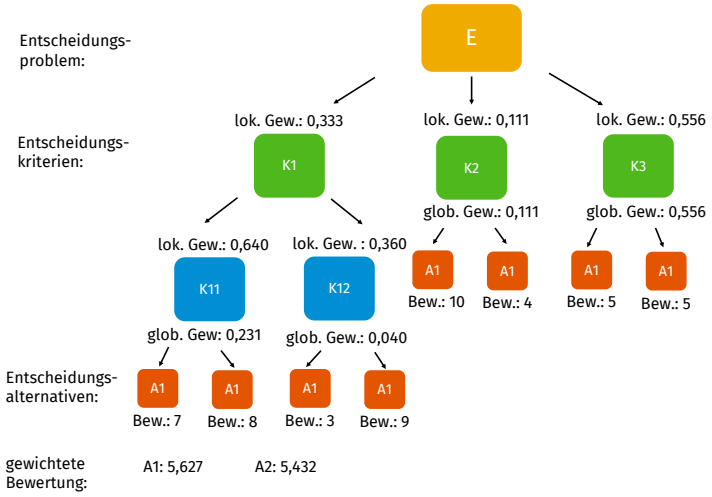

Abbildung 10: Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP. Eigene Darstellung.

Das Rahmenwerk eignet sich aufgrund des multidimensionalen Entscheidungsmodells besonders für die in der Arbeit zu untersuchende Fragestellung. Die bei der Wahl einer CI/CD-Pipeline zu berücksichtigenden Aspekte können somit als Bewertungskriterien in den AHP-Baum aufgenommen werden. Des Weiteren ermöglicht die im AHP-Verfahren abgewickelte Gewichtung, dass die Kriterien in unterschiedlichem Maße Einfluss auf die Bewertung einer Pipeline nehmen. Im Rahmen dieser Arbeit kann somit die Wichtigkeit der an die Entscheidung geknüpften Aspekte durch verschiedene an der Bereitstellung von Software beteiligten Stakeholder (Entwickler, DevOps-Spezialisten etc.) festgelegt werden. Dies ermöglicht eine auf die Präferenzen der Entscheidungsträger abgestimmte Bewertung der zu untersuchenden Pipelines. Während bei Methoden wie der SWOT-Analyse ausschließlich qualitative Aspekte berücksichtigt werden, ermöglicht das AHP-Model ebenfalls eine Evaluation quantitativer Bewertungsmetriken. Infolgedessen kann bei der Definition der Bewertungsmetrik für jedes Entscheidungskriterium überprüft werden, ob eine quantitative oder qualitative Bewertung geeignet ist.
Weitere Erläuterungen zu den im AHP-Verfahren getroffenen Entscheidungen werden zur besseren Verständlichkeit in Kapitel 4.2 ausgeführt.

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

Als Entscheidungsalternativen werden CI/CD-Pipelines für die Bereitstellung von Cloud-Software gegenübergestellt. Konkret handelt es sich dabei, um die Tools *SAP CI/CD*, *Jenkins*, sowie *Azure Pipelines*. Die Gegenüberstellung wird auf die genannten CI/CD-Pipelines beschränkt, da diese von der SAP als Best Practice definiert wurden. Bei der Untersuchung der Pipelines sind dabei verschiedene Aspekte zu beachten. So soll im Rahmen der Arbeit evaluiert werden, ob sich die verschiedenen Pipelines zur Bereitstellung von Software für CEs eignet. Darüber hinaus muss festgestellt werden, inwiefern die Tools für die Technologien SAP CAP bzw. SAP UI5 geeignet sind und ob eine Bereitstellung in der Laufzeitumgebung Cloud Foundry der SAP BTP möglich ist.

4.2.1 Festlegung der AHP-Entscheidungsalternativen

Die zur Durchführung des AHP-Verfahrens benötigten Daten werden neben einer Literaturrecherche ebenfalls mittels Experteninterviews erhoben. Dabei wird eine Expertengruppe aus X Mitarbeitenden der SAP zusammengestellt. Diese sind jeweils in spezifischen Bereichen der Cloud-Fullstack-Entwicklung, Test-Management, sowie im DevOps spezialisiert. Somit kann Expertise über verschiedene Fachbereiche hinweg aufgebaut und Anforderungen aller an der Entwicklung, Bereitstellung sowie den Betrieb von Software beteiligten Stakeholdern erfasst werden. Zur Festlegung der Entscheidungskriterien wird eine induktive Kodierung der Expertengespräche durchgeführt. Dabei werden aus besonders häufig von Experten genannten

Aspekten systematisch Kategorien abgeleitet. Diese umfassen insbesondere Aspekte, welche die in vergangenen Entwicklungsprojekten hervorgegangenen Anforderungen an eine CI/CD-Pipeline darstellen. Da innerhalb dieser Kundenprojekte oft weniger strikte Qualitätsanforderungen an den CI/CD-Prozess gestellt werden, wird darüber hinaus erarbeitet, welche internen Bestimmungen von der SAP zur Bereitstellung von Standardsoftware definiert werden. Die bei der induktiven Kodierung erhobenen Kategorien werden anschließend ebenfalls als Entscheidungskriterien im AHP-Verfahren wiederverwendet. Die mit dieser Vorgehensweise erhobenen Entscheidungsalternativen sind folgender Abbildung zu entnehmen:

Abbildung 11: AHP-Entscheidungsstruktur zur Bewertung von CI/CD-Pipelines. Eigene Darstellung.

Auf der obersten Ebene des AHP-Entscheidungsbaums werden neun Kategorien definiert. Das erste Kriterium ist **Funktionalität** (K1). Diese Kategorie umfasst verschiedene innerhalb des CI/CD-Workflows benötigte funktionale Spezifikationen. So sollte eine Pipeline etwa dazu in der Lage sein, Anwendungen zu testen,

Code-Analysen durchzuführen und eine Software auf der Cloud-Plattform bereitzustellen. Experte 1 begründet dabei, dass es „eine Unterstützung dieser Stufen benötigt, um eine effiziente Bereitstellung von Software zu ermöglichen.“ Angesichts der Vielfältigkeit des Entscheidungskriteriums Funktionalität, wird eine Untergliederung in verschiedene Subkriterien vorgenommen. In Kategorie 1.1 wird evaluiert, welche Tests von der Pipeline unterstützt werden. Von der SAP werden diesbezüglich Produktstandards vorgegeben. Diese stützen sich auf *ISO 9001*, eine internationale Norm für Qualitätsstandards. Im Kontext der Softwareentwicklung verlangt diese, eine für neue Funktionalität kontinuierliche und automatisierte durchgeführte Prüfung. Hinsichtlich der Entwicklung von CAP-Node-Anwendungen schreibt die SAP in ihren Produktstandards die Durchführung von Unit-Tests mittels Jest bzw. Mocha und Integration-Tests sowie Functional-Tests mittels Newmann vor. Für die Programmierung mit SAP UI5 sind hingegen Unit-Tests mittels Q-Unit, Integration sowie Functional-Tests mittels OPA5 und E2E-Tests mittels WDI5 vorgesehen. Während die Test-Frameworks Jest, Mocha, Newmann sowie Q-Unit in einer herkömmlichen Node-Laufzeitumgebung ausgeführt werden, benötigt es für OPA5 sowie WDI5 einer emulierten Browser-Umgebung. Die von dem Emulator generierte Benutzeroberfläche ermöglicht das Simulieren und Testen von gezielten Anwenderinteraktionen. Diese können z.B. das Ausfüllen von Formularen bzw. das Klicken auf Schaltflächen darstellen. Bei der Bewertung soll deshalb insbesondere evaluiert werden, ob neben der Node-Laufzeitumgebung ebenfalls eine emulierte Browser-Umgebung von den zu vergleichenden Pipelines unterstützt wird.

In Kriterium K1.2 wird die Kompatibilität verschiedener *Code-Analyse-Tools* untersucht. Es wurde gezielt eine Trennung dieser Kategorie mit dem Kriterium K1.2 (Tests) vorgenommen. Während Tests eine funktionale Erfüllung der Anforderung evaluieren werden mit Code-Analysen Code-Qualitätsstandards der entwickelten Features untersucht. Dabei wird evaluiert, ob statische Codeanalysen, Security- sowie Performance-Überprüfungen von den CI/CD-Pipelines unterstützt werden. Gemäß den Produktstandards der SAP sind für statische Code-Analysen Lint sowie SonarQube vorgeschrieben. Mit diesen Tools können neben syntaktischen Form-

qualitätsüberprüfungen ebenfalls Code-Metriken wie Komplexität oder Quellcode-Duplikate analysiert werden. Zur Durchführung von Sicherheitsüberprüfungen wird bei SAP CAP-Node-Anwendungen Checkmarx verwendet, während bei SAP UI5 DASTER zum Einsatz kommt. Die Unterstützung solcher Sicherheits-Tools ist dabei insbesondere für eine CEA essenziell. Die Verwendung von APIs zur Kommunikation zwischen einzelnen Microservices hat zur Folge, dass eine für unautorisierte Zugriffe begünstigte Angriffsfläche entsteht. Während für Performance-Tests von der SAP das Tool JMeter vorgeschlagen wird, gibt es bezüglich der Code-Coverage-Checks keine spezifische Vorgabe.

In der Kategorie K1.3 werden die *Build-Funktionalitäten* der CI/CD-Pipelines evaluiert. Um Anwendungen in die Cloud Foundry Laufzeitumgebung der SAP BTP bereitzustellen wird i.d.R. das Mulit-Target-Application-Konzept (MTA-Konzept) verwendet.

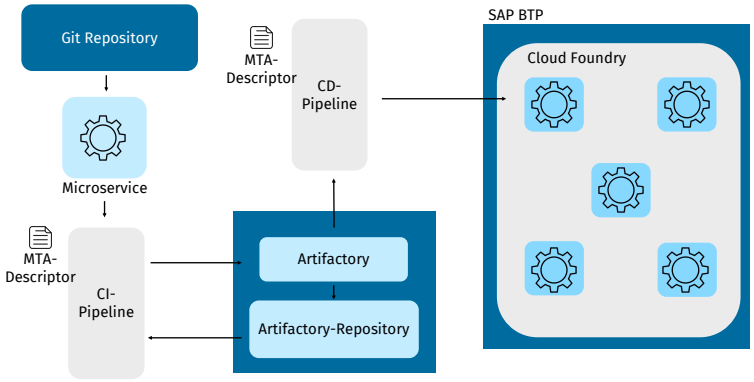

Abbildung 12: Bereitstellung von MTA-Applikationen in ein Artefakt-Repository. Eigene Darstellung.

Die MTA ist eine Applikation, z.B. ein Microservice eines CEs, welche aus verschiedenen Modulen besteht. Diese Module umfassen typischerweise die durch einen Microservice bereitgestellte API oder eine von der Applikation verwendete Datenbank sein. Eine CE-Anwendung kann dabei ebenfalls von verschiedenen externen Ressourcen abhängig sein. Diese sind in einem Artefakt-Repository verwaltete externe Komponenten, welche bei der Entwicklung neuer Microservices wiederverwendet werden können. Diese Komponenten werden während des Build-Prozesses von

der CI/CD-Pipeline aus den Artefakt-Repositorys geladen. Darüber hinaus können die von einer CI/CD-Pipeline bereitgestellten Anwendungen ebenfalls als wiederverwendbare Komponenten in das Artefakt-Repository eingelagert werden. Für CEA ist vorteilhaft, wenn verwendete CI/CD-Pipelines Docker-Workflows unterstützen. Durch den Einsatz von Docker-Containern können Entwickler schnell virtualisierte Umgebungen mit benötigten Frameworks und Tools bereitstellen ohne dabei eine gesamte Infrastruktur manuell konfigurieren zu müssen. Innerhalb dieses Bewertungskriteriums wird deshalb evaluiert ob, Build-Tools für MTA, Artefakt-Repositorys sowie Docker-Workflows durch die CI/CD-Pipelines unterstützt werden.
In Kriterium K1.4 werden die *Deploy- und Release-Prozesse* der CI/CD-Tools untersucht. Dabei wird evaluiert, ob die Pipelines verschiedene Bereitstellungsstrategien (z.B. Blue/Green-Deployment s. Kap. 2.2.3) unterstützen. Diese Strategien gewährleisten die notwendige Flexibilität und Agilität, welche es in CEs benötigt. Auf diese Weise können neue Anwendungsversionen schnell und ohne Beeinträchtigung der Produktivsysteme bereitgestellt und getestet werden. Des Weiteren wird erörtert, ob die CI/CD-Pipelines eine Bereitstellung in das SAP Cloud Transport Management (SAP CTM) unterstützten.

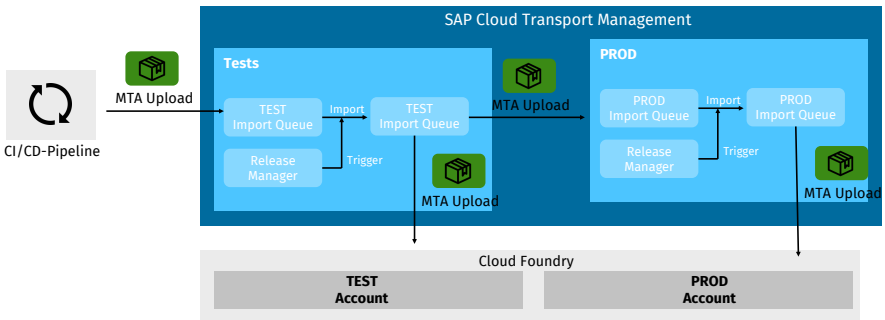

Abbildung 13: SAP Cloud Transportmanagement. Eigene Darstellung.

Das SAP CTM kann als zusätzliche Schicht im CI/CD-Prozess verbaut werden (s. Abb. 13). Mit dem SAP CTM können Anwendungsartefakte zwischen verschiedenen Accounts der SAP BTP verschoben werden. So könnte ein Unternehmen etwa einen Account für das Testen (TEST) bzw. für die Produktionsumgebung (PROD) besit-

zen. Nachdem der Entwickler seine Funktionalitäten fertiggestellt hat, kann das entsprechende Artefakt für letzte die manuellen Tests der Qualitätssicherung unmittelbar in die TEST-Umgebung bereitgestellt werden. Die Bereitstellung in die PROD-Umgebung erfolgt dabei unter Verwendung des SAP CTM durch das Betriebsteam. Dabei stellt das SAP CTM diverse Funktionalitäten, darunter eine eine zeitplangesteuerte Bereitstellung sowie das Definieren von Abhängigkeiten zu anderen Microservices, bereit. In Kategorie K1.5 wird die *Monitoring-Funktionalität* der verschiedenen CI/CD-Pipelines untersucht. In diesem Zusammenhang erfolgt eine Bewertung der Überwachbarkeit der CI/CD-Tools. So soll der fehlerfreie Bereitstellungs-Workflow etwa anhand von Logs oder Metriken wie Build-Zeiten evaluiert werden. Zusätzlich sollte es möglich sein, die Abwicklung der Tests sowie die Bereitstellung in die Produktionsumgebung zu überwachen.

In Kategorie K2 werden die **Integrationsmöglichkeiten** der Pipeline untersucht. In dem Subkriterium *Integrationsmöglichkeiten von Repositorys* (Kriterium K2.1) wird evaluiert, ob sich das Repository in die Pipeline integrieren lässt. Dies ist beispielsweise über Webhooks möglich. Mit diesen können bestimmte Ereignisse, wie das Pushen von Code-Änderungen in einem Repository automatisch an die CI/CD-Pipeline gesendet werden. Somit kann ein unmittelbarer Integrations- bzw. Bereitstellungs-Workflow ausgelöst werden. Bei der Bewertung wird dabei insbesondere darauf geachtet, dass häufig verwendete Repositorys in die Pipeline integrierbar sind. In Kriterium K1.2 werden die *Integrationsmöglichkeiten von Entwicklungsumgebung* untersucht. Die Integration-Pipeline kann unmittelbar während des Entwicklungsprozesses aus der Entwicklungsumgebung gestartet werden. Auf diese Weise wird sichergestellt, dass Entwickler Feedback in noch kürzerer Zeit erhalten als bei einer ausschließlichen Integration der Pipeline in das Repository. Die Bewertung bezieht sich dabei ausschließlich auf SAP UI5 sowie SAP CAP Entwicklungsumgebungen. Dazu gehören Microsoft Visual Studio Code, SAP Business Application Studio (SAP BAS) sowie Eclipse. In Kriterium K2.3 wird die *Integrationsmöglichkeiten von Planungssoftware* untersucht. Dazu gehören Projektmanagement-Tools wie Jira. Eine Integration solcher Planungssoftware erlaubt Projektmanager eine erhöhte Trans-

parenz über den Bereitstellungs-Workflow aller zu implementierender Arbeitselemente zu erlangen. Auf diese Weise kann der CI/CD-Status eines Backlog-Items unmittelbar über die Planungssoftware eingesehen werden. Da keine SAP-spezifischen Vorgaben bezüglich Planungssoftware getroffen sind, erfolgt lediglich eine Untersuchung der generellen Integrationsfähigkeit von Planungssoftware.
In Kriterium K3 erfolgt die Evaluation der **Kosten**. Um eine Vergleichbarkeit herzustellen, wird eine Analyse der Kosten pro Build-Stunde durchgeführt. Da die Installation und Wartung von Jenkins auf einem eigenen Server mit zusätzlichen Kosten verbunden ist, wird eine Bewertung der Kosten für eine in der Cloud betriebene Jenkins-Instanz durchgeführt. Hierfür wird der von SAP Hyperspace verwaltete Jenkins-as-a-Service (JaaS) als Vergleichsgrundlage verwendet.
Das Kriterium K4 untersucht die *Skalierbarkeit* der CI/CD-Pipelines. Hierbei werden die Pipelines auf horizontale so wie vertikale Skalierbarkeit untersucht. Die horizontale Skalierbarkeit ermöglicht eine parallele Durchführung mehrerer Builds. Gerade bei einer hohen Anzahl gleichzeitiger Hauptzweigintegrationen birgt dies einen hohen Mehrwert. Die vertikale Skalierung bezieht sich auf die Erhöhung der Ressourcen einer Pipeline-Instanz. So kann die CI/CD-Pipeline dynamisch an die sich ändernden Anforderungen eines angepasst werden. Insbesondere für CEs kann dies einen hohen Mehrwert darstellen. Durch schnelle und effiziente Entwicklungs- und Bereitstellugnsprozesse können diese Unternehmen das Time-to-Market verkürzen und somit schneller auf die Bedürfnisse der Kunden reagieren.
In Kriterium K5 wird die *Performance* der verschiedenen CI/CD-Pipeline verglichen. Dabei werden die zu untersuchenden Tools anhand derselben Anwendung getestet. Damit soll für jede Pipeline die zur Prozessierung des CI/CD-Workflows benötigte Zeit gemessen werden. Im Rahmen dieser Gegenüberstellung wird eine Unterscheidung zwischen der Integration- bzw. Delivery-Zeit realisiert. Die Integration-Zeit bezeichnet den Zeitraum, welcher von der Einführung eines Feature-Branchs bis zur vollständigen Konsolidierung in den Hauptzweig benötigt wird. Dabei werden in die Pipelines dem CI-Prozess entsprechende Validierungen wie Unit- und Integration-Tests eingebaut. Die Delivery-Zeit beschreibt die Zeitspanne, welche von der Frei-

gabe des Hauptzweigs bis zur Bereitstellung der Software auf die Cloud-Plattform benötigt wird. Dabei werden CD-typische Schritte in die Pipelines integriert. Dazu gehört das Ausführen von Code-Analysen und E2E-Tests.

In Kriterium K6 wird die **Flexibilität** der verschiedenen Pipelines evaluiert. Eine bedeutende Dimension der Flexibilität ist die uneingeschränkte Konfigurierbarkeit der Pipelines. So sollte eine Pipeline etwa keinerlei Beschränkungen in Bezug auf Anzahl und Reihenfolge der im CI/CD-Workflow durchzuführenden Schritten besitzen. Weiterhin wird evaluiert, ob für die Pipeline ein modularer Aufbau möglich ist. Da CEs i.d.R. über eine Vielzahl an Services verfügen, bedarf es ebenfalls einer hohen Anzahl an Pipelines. Um die daraus resultierende Komplexität zu reduzieren, sollten Pipelines aus modularen wiederverwendbaren Komponenten bestehen. Wird ein neuer Service in die Systemlandschaft integriert, können diese Komponenten somit ohne hohen Aufwand wiederverwendet werden. Ein weiterer für die Flexibilität der Pipelines essenzieller Aspekt ist die Unterstützung von Plugins. Mit Plugins können ebenfalls nicht im Standard verfügbare Funktionen in die Pipeline integriert werden. Dadurch sind CEs in der Lage, agil auf sich ändernde Bedürfnisse zu reagieren und können somit unabhängig von dem mit der Pipeline ausgelieferten Standard operieren.

In Kriterium K7 wird der für die CI/CD-Pipelines bereitgestellte **Support** evaluiert. Hierfür werden die Subkriterien *Administrativer Support* sowie *Community-Support* gebildet. Im Hinblick auf den *Administrativen Support* wird geprüft, ob die Pipeline-Anbieter Unterstützung bei der Einrichtung, Konfiguration sowie Problembehebung der CI/CD-Tools bietet. Dies ist insbesondere dann hilfreich, wenn der Umgang mit den Pipelines einen hohen Grad an Expertise benötigt. Des Weiteren wird evaluiert, ob Schulungen sowie Informationsmaterial verfügbar sind. Ein weiterer wesentlicher Aspekt ist die Verfügbarkeit von Updates. Durch kontinuierliche Updates kann sichergestellt werden, dass die Pipeline stets auf dem neusten Stand der Technik ist. Im Kontext des *Community-Supports* wird geprüft, ob öffentliche Foren existieren, in welchen Anwender Fragen stellen und Probleme diskutieren können. Darüber hinaus wird evaluiert, inwiefern eine Community zur Erweiterung der Dokumentation

oder zur Entwicklung neuer Funktionalität beiträgt.

In dem Kriterium K8 wird die **Sicherheit** der CI/CD-Pipelines untersucht. Hierbei werden die Subkriterien *Authentifizierung und Autorisierung* bzw. *Sicherheitsarchitektur* gebildet. Im Kontext der *Authentifizierung und Autorisierung* wird evaluiert, ob eine CI/CD-Pipeline geeignete Sicherheitsmaßnahmen implementiert. Um unerwünschte Zugriffe zu vermeiden, sollte die Verwendung einer Pipeline ausschließlich über eine Nutzer-Passwort-Kennung möglich sein. Besonders vorteilhaft ist dabei die Einbindung zentralisierte Drittanbieter, wie der SAP Identity Provider oder GitHub. Damit kritische Konfiguration ausschließlich von Spezialisten vorgenommen werden, muss eine Pipeline ebenfalls Autentisierungskonzepte unterstützten. Dabei sollen Benutzern über die Implementierung von Rollen bestimmte Rechte eingeräumt werden können. Unter dem Aspekt der *Sicherheitsarchitektur* wird die Systemintegrität der Pipeline-Tools untersucht. Zu Erhöhung der Sicherheit ist es z.B. vorteilhaft, wenn CI/CD-Pipelines in isolierten Umgebung, wie z.B. Docker-Container oder virtualisierten Maschinen laufen. Entstehen in der CI/CD-Pipeline Lücken, können sich diese nicht unmittelbar auf andere Systeme ausweiten. Ein weiterer in diesem Kriterium evaluierter Aspekt ist die Ausfallsicherheit. Um sicherzustellen, dass neue Funktionalität bei der Integration kontinuierlich getestet und Software schnell an den Kunden bereitgestellt werden kann, sollten die CI/CD-Systeme stets hochverfügbar sein. In Kriterium K9 wird die **Benutzerfreundlichkeit** der CI/CD-Pipelines untersucht. Dafür wird eine Unterteilung in *Installation und Wartung* sowie in *Intuitive Bedienbarkeit* vorgenommen. Hinsichtlich des Kriteriums der *Installation und Wartung* ist es dabei besonders vorteilhaft, wenn das CI/CD-Tool nicht installiert werden muss, sondern unmittelbar als Service bereitgestellt wird. Um sicherzustellen, dass die Pipeline hochverfügbar ist, sollte das Bereitstellungssystem kontinuierlich gewartet werden. Wird die Wartung dabei als Dienstleistung von einem CI/CD-Plattformanbieter übernommen, kann ein Unternehmen die Fachkraft auf das Kerngeschäft und somit auf die Entwicklung neuer Services konzentrieren. Auch der für die Implementierung und Konfiguration der Pipelines benötigte Aufwand sollte so gering wie möglich sein (*intuitive Bedienbarkeit*).

Um die Abhängigkeit einer Abteilung von hochqualifizierten DevOps-Spezialisten zu veringern, kann es etwa von Vorteil sein, wenn Pipelines nicht mittels Programmiersprachen, sondern über intuitive Benutzeroberfläche konfigurierbar sind.

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] Kathleen Gerson und Sarah Damaske. *The science and art of interviewing*. eng. Gerson, Kathleen (VerfasserIn) Damaske, Sarah (VerfasserIn). New York, NY: Oxford University Press, 2021. 280 S. ISBN: 9780199324293. DOI: 10.1093/oso/9780199324286.001.0001.

[5] 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.

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

[7] 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.

[8] Achim Hildebrandt u. a. *Methodologie, Methoden, Forschungsdesign. Ein Lehrbuch für fortgeschrittene Studierende der Politikwissenschaft*. Jäckle, Sebastian (author) Wolf, Frieder (author) Heindl, Andreas (author). Wiesbaden: Springer Fachmedien Wiesbaden, 2015. ISBN: 978-3-531-18256-8. DOI: 10.1007/978-3-531-18993-2.

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

[10] 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.

[11] 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.

[12] 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.

[13] Thomas L. Saaty. „Decision making with the analytic hierarchy process". In: *International Journal of Services Sciences* 1.1 (2008), S. 83. ISSN: 1753-1446. DOI: 10.1504/IJSSCI.2008.017590.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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

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

[20] 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).

[21] 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).

[22] Steve Denning. *Beyond Agile Operations: How To Achieve The Holy Grail Of Strategic Agility*. Hrsg. von Forbes. 2017. URL: https://www.forbes.com/sites/stevedenning/2017/02/10/beyond-agile-operations-how-to-achieve-the-holy-grail-of-strategic-agility/?sh=712d37dc2b6a (besucht am 16. 03. 2023).

[23] 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).

[24] 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).

[25] 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).

[26] 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).

[27] 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).

[28] 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