



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

Abstract

Die Composable-Enterprise-Architektur (CEA) ist ein von dem Analystenhaus Gartner veröffentlichtes IT-Konzept, welches darauf abzielt, Agilität, Skalierbarkeit und Anpassungsfähigkeit in Organisationen zu fördern. Diese System-Architektur besteht aus unabhängigen, in sich geschlossenen Services, welche bei Bedarf erweitert, verändert oder ausgetauscht werden können. Um diese modularen Software-Bausteine effizient in Produktionssysteme bereitstellen zu können, bedarf es einer Continuous-Integration-and-Delivery-Pipeline (CI/CD-Pipeline). Diese automatisiert den Prozess der Integration und Bereitstellung und ermöglicht, dass Code-Änderungen zuverlässig in lauffähige Anwendungen umgesetzt werden. Ziel der vorliegenden Arbeit ist, zu evaluieren, welches CI/CD-Pipeline-Tool zur Automatisierung der Bereitstellungsprozesse für eine CEA den größten Mehrwert birgt. Konkret werden dabei die Tools Azure Pipelines, Jenkins sowie SAP CI/CD verglichen. Zur Beantwortung der vorliegenden Forschungsfrage wurde der Analytische Hierarchieprozess (AHP) durchgeführt. Dabei wurden die zu vergleichenden CI/CD-Tools anhand von neun gewichteten Kriterien verglichen. Die zur Durchführung des AHP-Verfahrens benötigten Daten wurden mittels semistrukturierter Leitfadeninterviews erhoben. Die Auswertung des AHP-Verfahrens veranschaulicht, dass Azure Pipelines als das optimale CI/CD-Tool angesehen werden kann. Dies ist insbesondere auf die hohe Flexibilität der Pipeline zurückzuführen. Da Azure Pipelines eine Cloud-Lösung ist, können die Rechenressourcen des Pipeline-Systems dynamisch an die spezifischen Anforderungen eines Entwicklungsteams angepasst werden. Zudem stellt Azure Pipelines essenzielle Funktionalitäten bereit, welche für das Bauen, Testen und Bereitstellen von auf SAP-Technologien basierenden Applikationen benötigt werden. Das im AHP-Verfahren ermittelte Resultat muss jedoch für gewisse Anwendungsfälle abgegrenzt werden. Demnach wird empfohlen, dass Teams mit geringer CI/CD-Erfahrung SAP CI/CD in Betracht ziehen sollten, wohingegen Entwicklungsabteilungen, welche ein hohes Maß an Kontrolle über das Pipeline-System anstreben, die Nutzung von Jenkins erwägen sollten.

Inhaltsverzeichnis

Selbstständigkeitserklärung	II
Abkürzungsverzeichnis	VI
Abbildungsverzeichnis	VIII
Tabellenverzeichnis	IX
1 Einleitung	1
1.1 Motivation und Problemstellung	1
1.2 Zielsetzung und Abgrenzung	2
1.3 Aufbau der Arbeit	3
2 Grundlagen und Begriffserklärungen	5
2.1 Die Composable-Enterprise-Architektur	5
2.1.1 Begriffserklärung und Abgrenzung	5
2.1.2 Technologische Konzepte des Composable-Enterprises	8
2.2 Integration und Bereitstellung einer Cloud-Anwendung	10
2.2.1 Agile und DevOps als moderne Softwareentwicklungskonzepte	10
2.2.2 Automatisierung der Integrations- und Bereitstellungsprozesse	12
2.2.3 Strategien zur Bereitstellung von Neuentwicklungen	18
3 Methodische Vorgehensweise	21
3.1 Semistrukturierte Leitfadeninterviews zur Erhebung qualitativer Daten	21
3.2 Evaluation von Integrations- und Bereitstellungs-Tools unter Anwen- dung des Analytischen Hierarchieprozesses	23
4 Evaluation der Integrations- und Bereitstellungs-Pipelines unter Anwendung des Analytischen Hierarchieprozesses	27
4.1 Definition der Entscheidungskriterien	27
4.2 Festlegung der Bewertungsmetriken	34
4.3 Ermittlung der Gewichtungsfaktoren	36

4.4	Bewertung der Entscheidungsalternativen	37
5	Entwicklung einer Handlungsempfehlung	53
6	Schlussbetrachtung	61
6.1	Fazit und kritische Reflexion	61
6.2	Ausblick auf zukünftige Entwicklungen	63
	Literaturverzeichnis	X
	Anhang	XVIII

Abkürzungsverzeichnis

XP	Extreme Programming
DevOps	Development & Operations
CI/CD	Continuous Integration and Continuous Delivery
CI	Continuous Integration
CD	Continuous Delivery
DoD	Definition of Done
E2E-Tests	End-to-End-Tests
PBC	Packaged-Business-Capability
CEA	Composable-Enterprise-Architektur
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
AHP	Analytischer Hierarchieprozess
SAP BTP	SAP Business Technology Plattform
MTA	Multi-Target Application
SAP CTM	SAP Cloud Transport Management
SAP BAS	SAP Business Application Studio

DAST	Dynamic Application Security Testing
SAST	Static Application Security Testing
ERP	Enterprise-Ressourcen-Planning
CRM	Customer-Relationship-Management
TCO	Total Cost of Owner Ship
SSO	Single-Sign-On
CIO	Chief Information Officer
SAP DTS	SAP Data Technology Services
KI	Künstliche Intelligenz
TTM	Time-To-Market

Abbildungsverzeichnis

1	Aufbau der Arbeit	4
2	Entstehung einer Composable-Enterprise-Architektur	5
3	Bestandteile eines Composable-ERP-Systems	6
4	Technische Realisierung der Composable-Enterprise-Architektur . . .	8
5	Exemplarische Abfolge eines agilen Entwicklungszykluses	10
6	Aktivitäten im CI/CD-Prozess	12
7	Integration der CI/CD-Pipeline mit dem Versionskontrollsystem . . .	13
8	Hierarchische Darstellung von Softwaretests	14
9	Strategien zur Bereitstellung von Software	19
10	Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP	23
11	AHP-Entscheidungsstruktur zur Bewertung von CI/CD-Pipelines . .	28
12	Qualitative Bewertungsmetrik für AHP	34
13	Quantitative Bewertungsmetrik für AHP	35
14	Gewichtungsfaktoren der AHP-Entscheidungskriterien	36
15	SAP Cloud Transportmanagement	40
16	Pipeline-Monitoring mit Kibana und Jenkins	41
17	CEA-Szenario für Performance-Tests	45
18	Modularer Aufbau einer CI/CD-Pipeline	47
19	Entscheidungsbaum für die Wahl eines CI/CD-Pipeline-Tools	59

Tabellenverzeichnis

1	Exemplarische Darstellung der Paarvergleichsmatrix im AHP	24
2	Integration- und Delivery-Zeit in Sekunden	46
3	Ergebnistabelle zum AHP	52
4	Korrelation von Vorlaufzeit und Fehlerrate bei der Bereitstellung . . .	55

1 Einleitung

1.1 Motivation und Problemstellung

In der heutigen Zeit erweist sich Flexibilität als eine immer bedeutendere Determinante für den Erfolg von Unternehmen. So müssen diese stets in der Lage sein, sich effizient an verändernde Marktbedingungen anzupassen. Unternehmen sehen sich deshalb vornehmlich damit konfrontiert, die in Enterprise-Resource-Planning-Systemen (ERP-Systemen) abgebildeten Prozesse auf externe Einflüsse, wie Kundenbedürfnisse, rechtliche Vorschriften und neue Technologien auszurichten. Mit der *Composable-Enterprise-Architektur (CEA)* hat sich in den vergangenen Jahren ein Konzept etabliert, welches Unternehmen bei der Bewältigung dieser Disruptionen unterstützen soll. In einer CEA werden modulare Software-Komponenten zu einem homogenen Gesamtsystem konsolidiert [42]. Martin Henning, Head of New Ventures and Technologies bei der SAP, bemerkt, dass Unternehmen mit diesem Architekturkonzept nicht länger auf „rigide Systeme“ angewiesen, sondern vielmehr in der Lage sind, Geschäftsprozesse auf Grundlage einzelner Software-Bausteine zusammenzusetzen [38]. Bei einer Änderung der Geschäftsanforderungen besteht somit die Möglichkeit, einzelne Software-Komponenten dynamisch auszutauschen, ohne dass Anpassungen am Gesamtsystem erforderlich sind. Dieser Composable-Trend ist nicht nur bei der SAP, sondern ebenfalls bei anderen Unternehmen erkennbar. So wurde in Gartners Top-Trend-Forschung 2022 prognostiziert, dass bis zum Jahr 2024 80 Prozent der befragten Chief Information Officers (CIOs) die modulare Gestaltung von Geschäftsprozessen als eine der fünf wichtigsten Gründe für betriebliches Wachstum betrachtet werden [45]. Um die Geschäftsprozesse in einer CEA noch individueller auf die eigenen Bedürfnisse zuschneiden zu können, neigen Unternehmen dazu, die bestehende Architektur um eigenentwickelte modulare Bausteine zu erweitern. Damit Effizienz und Anpassungsfähigkeit vollständig ausgeschöpft werden kann, ist es unerlässlich, dass diese Bausteine schnell bereitgestellt und in das bestehende System integriert werden. Abhilfe schaffen soll dabei die in der Literatur als *Continuous Integration and Continuous Delivery (CI/CD)* bekannte Entwickler-

praktik. Damit soll sichergestellt werden, dass Änderungen am Code häufiger und zuverlässiger in den Produktionssystemen bereitgestellt werden. Dies kann mithilfe einer CI/CD-Pipeline realisiert werden. Mit dieser ist es möglich, die in einer Entwicklungsabteilung anfallenden Software-Bereitstellungsprozesse vollständig zu automatisieren. Der *State of DevOps Report* zeigt, dass die in Bereitstellungssystemen abgewickelten Tests sowie das zyklische Ausliefern von Software zu einem signifikanten Rückgang der Ausfallrate in Produktionssystemen geführt hat [35, S. 10]. Da CI/CD-Pipelines eine kontinuierliche Bereitstellung und Integration modularer Software-Komponenten ermöglichen, sind diese insbesondere für CEAs von großer Bedeutung. Um den spezifischen, in einer CEA vorliegenden Bedürfnissen gerecht zu werden, benötigt die Implementierung von CI/CD-Prozessen jedoch im Vergleich zu herkömmlichen System-Architekturen einer differenzierten Herangehensweise. Damit die Unabhängigkeit einzelner Software-Komponenten gewährleistet werden kann, besteht für CEA die Notwendigkeit einer granularen und dezentralen Pipeline-Struktur. Dabei bleibt zu hinterfragen, ob gegenwärtige CI/CD-Tools in der Lage sind, diesen Anforderungen zu genügen.

1.2 Zielsetzung und Abgrenzung

Die technische Beratungsabteilung SAP Data Technology Service (SAP DTS) unterstützt Kunden bei der Implementierung einer CEA auf der SAP-Cloud-Plattform. Ein essenzielles Thema stellt in diesem Kontext ebenfalls die Beratung bei der Implementierung von Software-Bereitstellungsprozessen dar. Um diese Vorgänge zu automatisieren, werden von dem SAP DTS i.d.R. drei verschiedene CI/CD-Pipeline-Tools empfohlen. Dazu gehören Azure Pipelines, Jenkins und SAP CI/CD. Ziel der Arbeit ist deshalb, zu evaluieren, welches dieser Tools den größten Mehrwert zur Bereitstellung von Cloud-Software für eine CEA liefert. Dafür soll ein Entscheidungs-Framework erstellt werden, anhand dessen eine Bewertung der Lösungen erfolgt. Es besteht die Möglichkeit, dass das mit dem Entscheidungs-Framework erhaltene Resultat nicht auf alle Projektkontexte übertragbar ist. Aus diesem Grund ist in der vorliegenden Arbeit ebenfalls vorgesehen, das Ergebnis in einer abschließen-

den Handlungsempfehlung einzuordnen und abzugrenzen. Daraus resultiert folgende Forschungsfrage:

Welches Tool bietet zur Automatisierung der Bereitstellungsprozesse für Composable-Enterprise-Architekturen den größten Mehrwert?

Im Rahmen der Zielsetzung werden folgende Abgrenzungen vorgenommen: Auf der SAP-Cloud-Plattform können Anwendungen auf verschiedenen Laufzeitumgebungen betrieben werden. Dafür wird neben Neo und Kyma ebenfalls Cloud Foundry bereitgestellt. In der vorliegenden Arbeit wird dabei jedoch ausschließlich die Bereitstellung von Software in der Cloud-Foundry-Laufzeitumgebung untersucht. Zudem beschränkt sich die Analyse der CI/CD-Tools auf Anwendungen, welche auf den Programmier-Frameworks SAP CAP Node sowie SAP UI5 basieren. Diese Abgrenzung wird gezogen, da das SAP DTS ausschließlich in diesen Technologien berät.

1.3 Aufbau der Arbeit

Die theoretischen Grundlagen beginnen mit der Begriffsdefinition und Abgrenzung der CEA. In diesem Zusammenhang werden sowohl betriebswirtschaftliche Prinzipien als auch die mit der IT-Architektur herbeigeführten Unternehmenspotenziale veranschaulicht. Im Anschluss werden technologische Konzepte der CEA erläutert. Dabei wird dargelegt, wie Composable-Enterprises (CEs) ihre Architektur gestalten müssen, um geschäftlichen Abläufe adäquat in der Unternehmenssoftware abzubilden. Im nachfolgenden Abschnitt werden im Rahmen der Softwareentwicklung anfallende Integrations- und Bereitstellungsprozesse beschrieben. Hierbei wird zunächst erläutert, wie die Entwicklungskonzepte *Agile* und *DevOps* zur Optimierung dieser Prozesse beitragen. Daraufhin werden CI/CD-Pipelines, also Tools zur Automatisierung dieser Bereitstellungsprozesse, erläutert. In diesem Kontext werden insbesondere die dabei anfallenden Prozesse, bei der SAP verwendete Tools und deren Integration in den Softwareentwicklungsprozess fokussiert. Im Methodikteil wird das gewählte Vorgehen zu den Experteninterviews, welche zur Erhebung qualitativer Daten durchgeführt werden, erläutert. Des Weiteren wird der *Analytische Hierarchieprozess (AHP)*, das Instrument zur Bestimmung des optimalen CI/CD-

Tools beschrieben. Im Teil der Durchführung erfolgt die Anwendung der Methodik auf die in den Experteninterviews erhobenen Daten. So werden im Rahmen des AHP-Verfahrens Entscheidungskriterien festgelegt und gewichtet. Um eine objektive Evaluierung der CI/CD-Pipeline-Tools zu ermöglichen, werden in diesem Abschnitt ebenfalls Bewertungsmetriken definiert. Im Anschluss werden die zu untersuchenden Entscheidungsalternativen anhand jedes Kriteriums bewertet. Auf dieser Grundlage lässt sich ein aggregierter Nutzwert ermitteln, welcher wiederum zur Identifikation des optimalen CI/CD-Tools beiträgt.

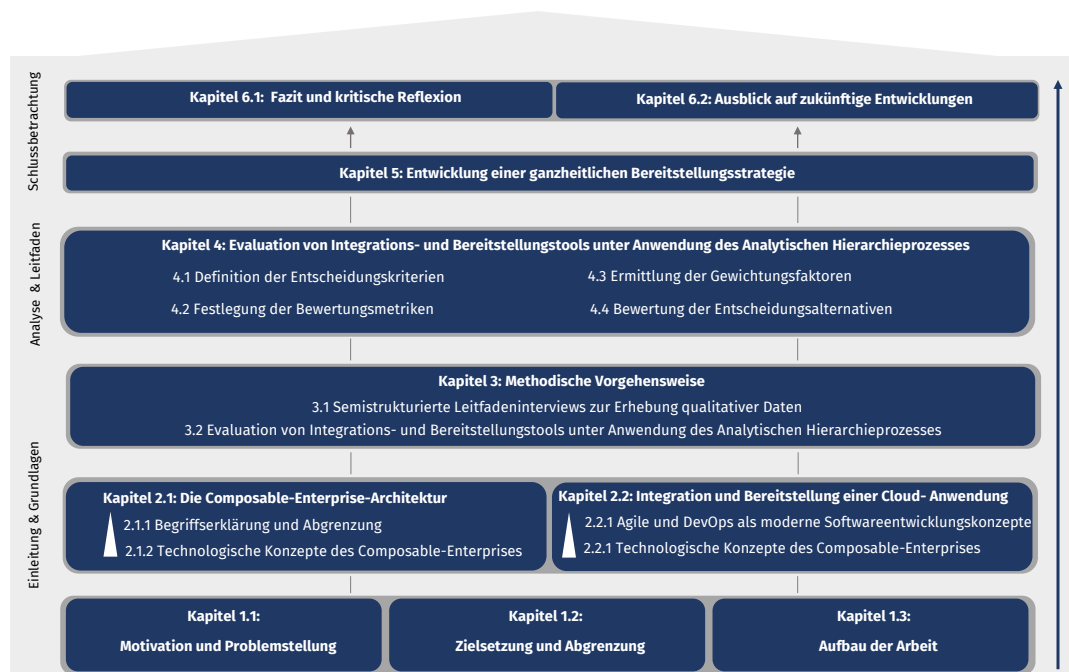


Abbildung 1: Aufbau der Arbeit. Eigene Darstellung.

In der folgenden Handlungsempfehlung wird eine ganzheitliche Bereitstellungsstrategie für Unternehmen, welche eine CEA implementieren entwickelt. Dabei wird das Ergebnis des AHP-Verfahrens analysiert und in Abhängigkeit verschiedener Unternehmensstrategien abgegrenzt. Abgerundet wird die Arbeit durch die Zusammenfassung der Erkenntnisse, einer kritischen Beleuchtung des Vorgehens und der Darstellung zukünftiger Forschungsansätze.

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, stellen diese Eigenschaften wesentliche Faktoren dar, welcher zur Steigerung der Wettbewerbsfähigkeit von Unternehmen beitragen [34]. Für Analystenhäuser wie Gartner steht 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-Architektur (CEA)**.

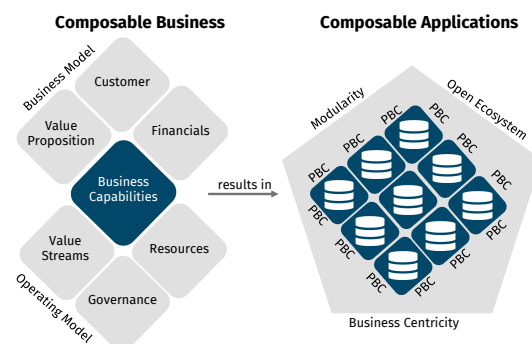


Abbildung 2: Entstehung einer Composable-Enterprise-Architektur. In Anlehnung an Schönstein [52].

In der Literatur wird die CEA dabei wie folgt definiert:

„The Composable Enterprise Architecture (CEA) is an approach to designing and implementing enterprise architectures focused on flexibility, scalability, and adaptability. It does this through the assembly and combination of packaged business capabilities [39].“

Ein CE 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. 2). Das Konzept

der PBCs fußt dabei auf den drei Prinzipien der CEA: *Modulare Architektur*, *Offenes Ökosystem* und *Businesszentriertheit* [42]. 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“ [42]. In diesem Kontext wird von dem Analystenhaus Gartner ebenfalls der Begriff des *Composable-Enterprise-Ressourcen-Planning-Systems (Composable-ERP-Systems)* verwendet. Hierbei stellen die modularen Komponenten (PBCs) vorgefertigte Geschäftsfunktionen- bzw. -prozesse dar, welche als Module in ein Composable-ERP-System integriert werden können. Diese PBCs können etwa Funktionen wie Finanzbuchhaltung, Einkauf, Verkauf, Lagerverwaltung, Produktion oder Personalmanagement enthalten. Ergibt sich eine Änderung in den Geschäftsanforderungen, ermöglicht diese Architektur ein flexibles und isoliertes Austauschen, Verändern sowie Weiterentwickeln einzelner PBCs [39]. Durch die unabhängige Bereitstellung der Softwarekomponenten ist es weiterhin möglich, einzelne Module einer CEA skalierbar zu gestalten, ohne dabei die Gesamt-Suite anpassen zu müssen. Insbesondere für Start-ups ist dieser Vorteil von hoher Bedeutung, da diese somit in der Lage sind, Unternehmenssoftware flexibel an wachsenden Geschäftsanforderungen anzupassen [15, S. 7].

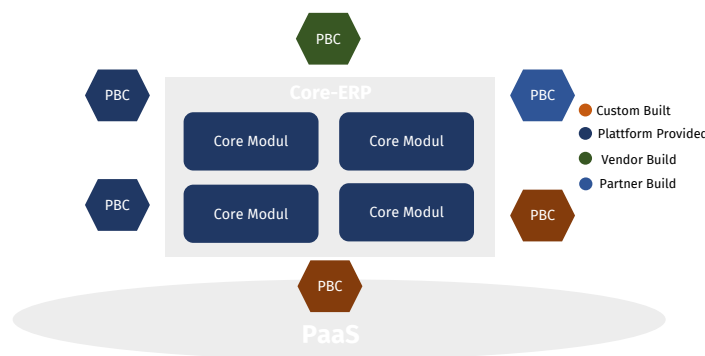


Abbildung 3: Bestandteile eines Composable-ERP-Systems. Eigene Darstellung.

Ein Composable-ERP-System verfügt über eine Kollektion von Kernkomponenten (s. Abb. 3). Diese Kernkomponenten werden i.d.R. von einem einzigen ERP-Anbieter bereitgestellt und können deshalb ohne hohen Aufwand integriert und auf-

einander abgestimmt werden [29]. Bei diesen Komponenten handelt es sich dabei um Funktionalitäten, welche das Hauptgeschäft eines Unternehmens unterstützen. Um den sich ändernden Geschäftsanforderungen gerecht zu werden, können diese Kernkomponenten durch zusätzliche PBCs erweitert werden. Dieses Konzept basiert auf dem Prinzip des *offenen Ökosystems*. Die externen, in das Kern-ERP-System integrierbaren PBCs, umfassen dabei i.d.R. stark spezialisierte Funktionalitäten. So könnte sich ein E-Commerce-Unternehmen dazu entschließen, eine Kern-Customer-Relationship-Management-Komponente (Kern-CRM-Komponenten) um eine KI-basierte Kundenanalysefunktion zu erweitern. Um die Integration dieser modularen Komponenten zu erleichtern, stellen ERP-Anbieter auf ihren Cloud-Plattformen einen zur Bündelung der PBCs verwendeten Marktplatz zur Verfügung [29]. Die dabei angebotenen Komponenten können zusätzliche Funktionen des ERP-Kernanbieters oder externe Komponenten von Spezialherstellern darstellen. Auf diese Weise haben Mitarbeiter oder Teams die Möglichkeit, innerhalb ihres Unternehmens auf diesen Marktplatz zuzugreifen und Werkzeuge (PBCs), welche zur Unterstützung der operativen Tätigkeiten benötigt werden, ohne hohen Aufwand zu aktivieren. Das ERP-System kann somit auf die spezifischen Bedürfnisse und Anforderungen der Unternehmen zugeschnitten werden [42]. IT-Systeme dienen dem Zweck der Unterstützung operativer Aufgaben. Um eine anwenderzentrierte Gestaltung der auf dem Marktplatz angebotenen Werkzeuge zu ermöglichen, sollte der Fokus dieser Tools auf den Bedürfnissen und Erwartungen der Anwender liegen (*Businesszentriertheit*). Deshalb ist essenziell, dass Mitarbeiter Systeme intuitiv nutzen und ggf. weiterentwickeln und anpassen können, ohne dabei von IT-Abteilungen abhängig zu sein [42]. Dabei soll die Verwendung von Low-Code/No-Code-Plattformen Abhilfe schaffen. Diese ermöglichen den Mitarbeiter eigene PBCs zu entwickeln ohne auf spezielle IT-Kenntnisse oder -Ressourcen angewiesen zu sein. Damit wird die Agilität und Flexibilität der CEs erhöht, während die Abhängigkeit von IT-Abteilungen reduziert wird.

2.1.2 Technologische Konzepte des Composable-Enterprises

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

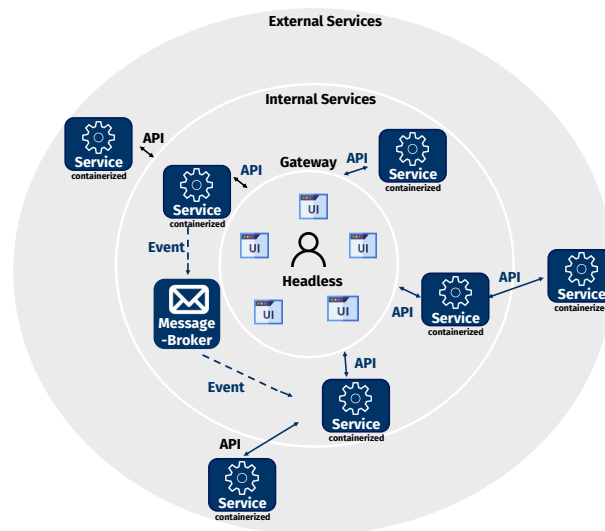


Abbildung 4: Technische Realisierung der Composable-Enterprise-Architektur. Eigene Darstellung.

Der zentrale Einstiegspunkt in eine CE-Anwendung ist das User-Interface. Insbesondere für Content-Management-Systeme wird für Frontend-Entwicklungen das *Headless-Konzept* verwendet (s. Abb. 4). 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 [42]. Die Geschäftsfunktionen einer CEA werden in verschiedene PBCs gekapselt. In der Applikationslogik des Backends kann ein PBC durch ein oder mehrere *Microservices* dargestellt werden. Ein Microservice ist eine eigenständige Einheit, welche für eine spezifische Funktion zuständig ist. Im Kontext eines E-Commerce-Unternehmens könnte ein PBC etwa eine Bestellverwaltung sein. Diese Bestellverwaltung kann dabei aus verschiedenen Microservices, wie einem Bestellannahme-, Auftragsabwicklungs- oder Rechnungstellungsdienst bestehen. Microservices spielen dabei hauptsächlich für die technische Realisierung eine Rolle. Folglich sind

diese für den Endanwender nicht erkennbar. Es ist möglich für jeden Service eine unterschiedliche Programmiersprache sowie Datenbank zu verwenden. So können Architektur und Technologien eines Dienstes unmittelbar an dessen betriebswirtschaftliche Anforderungen angepasst werden [11, S. 41]. 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. Ein weiteres in CEs verwendetes Kommunikationskonzept ist die *Event-driven Architecture (EDA)*. Die EDA ist ein Architekturkonzept, bei welchen Microservices asynchron über eine zentrale Vermittlungsinstanz, dem *Message Broker*, kommunizieren [2, S. 54]. 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 [13, S. 5]. Dieses Computing-Modell 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 [13, S. 10]. 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)* [13] [13, 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 Plattform (SAP BTP). Diese stellt eine Reihe von Diensten und Funktionen zur Verfügung, mit welchen Unternehmen SAP-ERP-Systeme anpassen, integrieren und erweitern können.

2.2 Integration und Bereitstellung einer Cloud-Anwendung

2.2.1 Agile und DevOps als moderne Softwareentwicklungskonzepte

Das Hauptaugenmerk eines CE 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 Projekt-elemente *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 [18, S. 5]. Dies resultiert nicht nur in einem Anstieg der Kosten, sondern führt ebenfalls dazu, dass IT-Projekte länger als geplant ausfallen [17, 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. 5) [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.

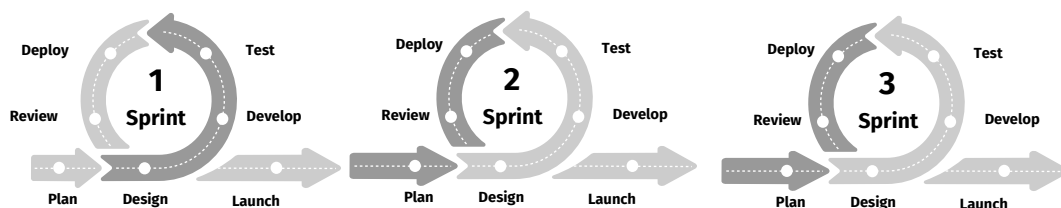


Abbildung 5: Exemplarische Abfolge eines agilen Entwicklungszykluses. In Anlehnung an K&C [20].

Sprints sind Durchläufe, welche i.d.R. einen Zeitraum von ein bis vier Wochen umfassen. Während dieses Abschnitts ist die Fertigstellung eines vor dem Sprint definierten Aufgabenkontingents (*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. Nach Ablauf eines Sprints kann das an die Stakeholder ausgelieferte Artefakt als Feedback-Grundlage verwendet und im unmittelbaren Folge-Sprint eingearbeitet werden [20, 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 [25]. Traditionell erfolgt eine funktionale Trennung der Entwickler- und IT-Betrieblerteams. Das Problem der letzten Meile beschreibt dabei, dass aufgrund ausbleibender Kooperation dieser Teams der Programmcode nicht auf die Produktivumgebung abgestimmt ist. Erkenntnisse aus der Praxis zeigen, dass solche organisatorischen Silos häufig in einer schlechten Software-Qualität und somit in einem geminderten Ertragspotenzial bzw. in einer Erhöhung der Betriebskosten resultieren [6, S. 1]. So geht aus der von McKinsey veröffentlichten 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 [43]. 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. Prägnant zusammenfassen lässt sich das DevOps-Konzept durch das Akronym CAMS: *Culture (Kultur)*, *Automation (Automatisierung)* und *Measurement (Messung)* [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, Disposi-

tionen 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 ist für Softwareentwicklungsunternehmen insbesondere der *Time-to-Market (TTM)* signifikant [6, S. 7]. Dieser beschreibt die Zeitspanne zwischen Entwicklungsprozess und der Markteinführung von IT-Services [16, S. 141]. Dabei soll die mit DevOps angestrebte Verschmelzung der Entwicklungs- und Betriebsteams, die Automatisierung von Prozessen sowie die kontinuierliche Integration und Bereitstellung zu einer erheblichen Reduzierung dieser Metrik führen. Adam Caplan, leitender Strategieberater bei Salesforce, sieht in einer schnellen Bereitstellung von IT-Services angesichts der bei Softwareintegration entstehenden Komplexität, einen erheblichen Vorteil. So können Anwendungen schnellst möglichst in produktionsähnlichen Umgebungen getestet und sukzessive mit dem Kunden verbessert werden [16].

2.2.2 Automatisierung der Integrations- und Bereitstellungsprozesse

Ein integraler Bestandteil des DevOps-Rahmenwerks ist *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 sog. CI/CD-Pipeline, welche alle Schritte von Code-Integration bis Bereitstellung der Software automatisiert. Hauptaugenmerk liegt dabei auf einer zuverlässigen und kontinuierlichen Bereitstellung von Software [19, S. 471].

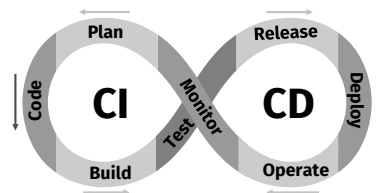


Abbildung 6: Aktivitäten im CI/CD-Prozess. In Anlehnung an Synopsys [56].

Alle in diesem Prozess anfallenden Aktivitäten werden dabei im CI/CD-Zyklus der Abb. 6 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 [19, S. 471]. Der erste Schritt des CI-Prozesses umfasst die Planung zu entwickelnder Services (*Plan*: s. Abb. 6). 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 Produktsystems abgestimmt ist, sollte zu jedem Zeitpunkt das Know-how der Betriebsteams einbezogen werden [6, S. 16]. Nach erfolgreichem Entwurf zu implementierender Anwendungs-Features beginnt die Entwicklung der IT-Services (*Code*: s. Abb. 6). Arbeiten hierbei mehrere Entwickler parallel an demselben IT-Service, wird der entsprechende Quellcode in Versionsverwaltungssysteme (*Repositories*) wie Github oder Bitbucket ausgelagert.

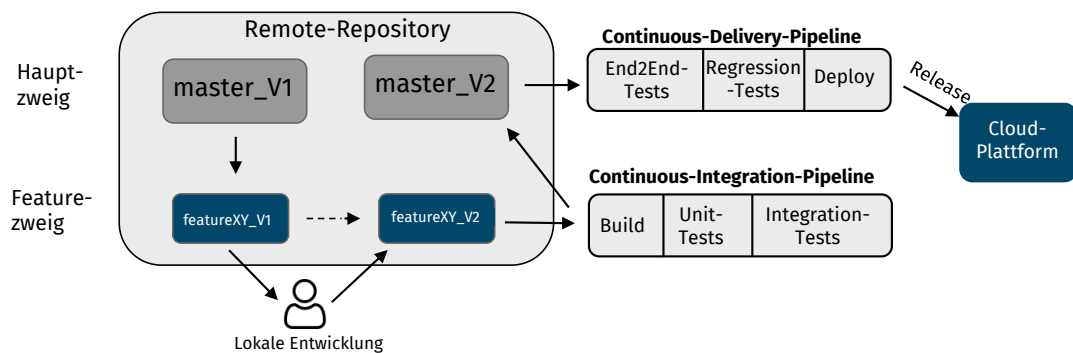


Abbildung 7: Integration der CI/CD-Pipeline mit dem Versionskontrollsystem. Eigene Darstellung.

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*) enthält eine aktuelle und funktionsfähige Version des Codes. Dieser mit verschiedenen Validierungsprozessen überprüfte Code stellt dabei die aktuelle in dem Produktionssystem laufende Anwendungsversion dar

(s. Abb. 7). Im Sinne der agilen Entwicklung werden große Softwareanforderungen (*Epics*), in kleine Funktionalitäten (*User Stories*) segmentiert, welche in separate Feature-Banches des Repositories 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 [10, S. 169]. Die Einbindung des Feature-Banches in den Hauptzweig resultiert i.d.R. in einem unmittelbaren und automatisierten Start des **CI/CD-Pipeline-Prozesses**. Bei der CI/CD-Pipeline handelt es sich dabei um ein vom Repository unabhängiges Bereitstellungsautomatisierungstool, 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. 6). 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*).

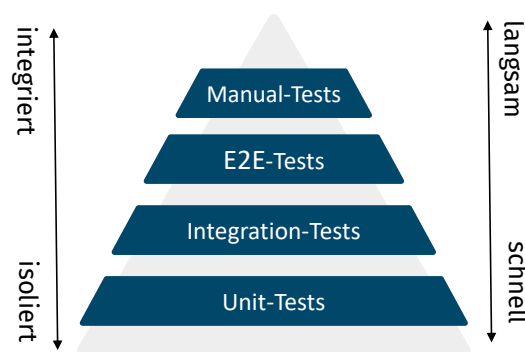


Abbildung 8: Hierarchische Darstellung von Softwaretests.
In Anlehnung an Paspelava [46].

Damit soll sichergestellt werden, dass zu jeder Zeit ein rudimentär getesteter Code bereitsteht und grundlegende Funktionalitäten sowie Schnittstellen erwartungsgemäß ausgeführt werden [6, S. 19]. Die in diesem Schritt abgewickelten Tests leiten sich dabei aus der *Definition of Done (DoD)* ab. Die DoD ist eine in der Planungs-

phase 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 Tests zu entwerfen (*Test Driven Development*). Der in dem CI-Prozess 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]. Unit-Tests befinden sich dabei auf unterster Hierarchieebene der Test-Pyramide (s. Abb. 8). 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 alle Tests erfolgreich absolviert wurden, kann sichergestellt werden, dass der neue Quellcode stabil, also funktionsfähig ist und keine Konflikte mit dem aktuellen Code des Hauptzweiges aufweist. Somit werden alle validierten Änderungen automatisch in dem 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 eingeschaltet [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-Systemkonfigurationen werden dabei so angelegt, dass diese der Produktionsumgebung möglichst ähnlich sind [9, Kap. 1.3]. Neben Datenbanken werden hierbei ebenfalls Serverkonfigurationen, wie Firewall- oder Netzwerkeinstellungen von dem Produktivsystem übernommen. Somit soll sichergestellt werden, dass eine neue Anwendungsversion unter produktionsähnlichen Bedingungen getestet wird. Analog zum CI-Prozess werden innerhalb des CD-Workflows ebenfalls Unit- und Integration-Tests abgewickelt. Im Gegensatz zur CI-Pipeline werden dabei ebenfalls rechenintensive Tests automatisiert. Somit werden im CD-Prozess essenzielle, jedoch während des Entwicklungsworkflows zu aufwendige Validierungen durchgeführt [6, S. 20]. Darüber hinaus werden in der Staging Area ebenfalls in der Test-Pyramide (s. Abb. 8) höher positionierte, also rechenintensivere Tests ausgeführt [7, Kap. 2]. Dazu gehören *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. Dabei wird i.d.R. eine Benutzeroberfläche emuliert mit welcher etwa das Anmelden mit Benutzername, das Suchen eines Produktes und das Abschließen einer Bestellung validiert werden kann [26]. Für kritische Systeme werden während des Delivery-Prozesses ebenfalls *Regression-Tests* vorgenommen. Diese umfassen ein erneutes Testen bereits ausgelieferter Softwarekomponenten. Regression-Tests können dabei in Form von Unit-, Integration- sowie Functional-Tests ausgeführt werden. Somit soll sichergestellt werden, dass sich Quellcodeänderungen nicht negativ auf die stabile

Anwendungsversion auswirkt. Auf oberster Ebene der Test-Pyramide befinden sich die *Manual-Tests*. Dabei handelt es sich um von menschlichen Testern ausgeführte Validierungen, mit welchen Benutzerfreundlichkeit sowie Funktionalität anhand authentischer Anwenderszenarien gewährleistet werden soll. Da diese Tests nicht automatisiert durchgeführt werden können, muss der nächste Schritt der CD-Pipeline nach erfolgreicher Validierung manuell angestoßen werden. Im Anschluss werden i.d.R. verschiedene Codeanalysen abgewickelt. 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. 6). 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. 6). Diese wird i.d.R. durch ein unabhängiges Überwachungssystem und nicht von dem Pipeline-Tool selbst abgewickelt. Dabei können z.B. Dashboards zur Analyse der Build-, Test- und Deployment-Prozesse visualisiert werden. Darüber hinaus umfasst dieses Tool essenzielle Überwachungselemente zum *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 Komponenten wie Datenbanken, virtuelle Netze bzw. Middlewares ordnungsgemäß durchgeführt werden. Das Anwendungs-Monitoring umfasst die Überwachung der Funktionalitäten und der Applikation selbst [6, S. 21].

Zur Automatisierung der CI/CD-Prozesse werden von der SAP i.d.R. drei verschiedene Tools vorgeschlagen. 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 und SAP CAP Node sowie der Laufzeitumgebung Cloud-Foundry kompatibel [62]. Eine wei-

tere bei der SAP empfohlene CI/CD-Alternative ist das Open-Source-Tool *Jenkins*. Im Gegensatz zum templatebasierten SAP CI/CD, muss der Bereitstellungswork-flow bei Jenkins mit der Programmiersprache Groovy implementiert werden. Weiterhin wird Jenkins nicht unmittelbar auf der SAP BTP betrieben, sondern muss auf einem eigenen Server (On-Premise) verwaltet werden [9, Kap. 2]. Um die Bereitstellung von SAP-spezifischen Technologien zu optimieren, wurde die Programmbibliothek *Project Piper* veröffentlicht. In Project Piper werden vorimplementierte Pipeline-Schritte und die dafür benötigten Treiber gebündelt. Im Gegensatz zum SAP CI/CD sind diese jedoch hoch konfigurierbaren. Ein externes ebenfalls von der SAP vorgeschlagenes 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-Plattform 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 Validierungen. Anhand der Bereitstellungsstrategie wird festgelegt, mit welcher Methode und zu welchem Zeitpunkt Nutzeranfragen von der aktuellen auf die neue Anwendungsinstanz umgeleitet werden (s. Abb. 9). Eine häufig verwendete Deployment-Strategie ist dabei das *Blue/Green-Deployment*. Hierbei wird neben der stabilen aktuellen Anwendung (*Blaue Version*) ebenfalls eine Instanz des neuen IT-Services (*Grüne Version*) in dem Produktionssystem betrieben. Das impliziert, dass der alte IT-Service nicht unmittelbar, sondern erst nach verschiedenen im Produktionssystem durchgeführten Validierungen durch die aktualisierte Version ersetzt wird. Ein Vorteil dieser Vorgehensweise besteht darin, dass die neue Version nicht in einer Staging-Area, sondern unmittelbar unter produktionsähnlichen Bedingungen

geprüft werden kann [59]. Ein zusätzlicher positiver Effekt entsteht in Zusammenhang mit der durch die Aktualisierung verursachten Ausfallzeit. Im herkömmlichen Bereitstellungsverfahren wird die alte Anwendungsversion gestoppt, der neue Service installiert und dann gestartet. Dadurch kann der Dienst erst nach erfolgreicher Inbetriebnahme der neuen Instanz wiederverwendet werden, was i.d.R. zu längeren Ausfallszeiten führt. Dies kann jedoch mithilfe des Blue/Green-Deployments vermieden werden. Hierbei wird die neue Anwendungsversion auf einer separaten Instanz installiert. Sobald die neue Version erfolgreich gestartet ist, kann der Datenverkehr unmittelbar auf diese umgeleitet werden. Somit können längere Ausfallszeiten umgangen werden.

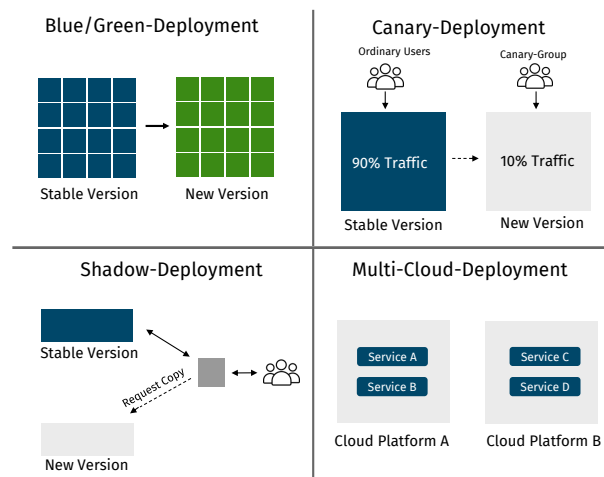


Abbildung 9: Strategien zur Bereitstellung von Software.
In Anlehnung an Ugochi [59].

Im Gegensatz dazu wird im Blue-Green-Deployment die neue Anwendung auf einer separaten Instanz installiert. Sobald die neue Version erfolgreich gestartet wurde, kann der Datenverkehr unmittelbar auf diese umgeleitet werden. Im Vergleich 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 Anwen-

dungen überprüft und ggf. Anpassungen vorgenommen werden. Bevor die neue Anwendungsversion der gesamten Nutzerbasis zur Verfügung gestellt wird, kann diese sukzessive und schrittweise ausgerollt werden [59]. 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 [59]. Ein weiteres Bereitstellungskonzept ist das *Multi-Cloud-Deployment*. Dieses wird verwendet, um ERP-Dienste in einer verteilten Computing-Umgebung auszuführen. Dabei werden die Anwendungen eines Unternehmens i.d.R. auf verschiedenen Cloud-Plattformen bereitgestellt. Der Vorteil dieser Bereitstellungsstrategie besteht darin, dass durch diese Diversifizierung verschiedene Technologien verwendet und somit eine höhere Skalierbarkeit, Verfügbarkeit und Performance der Anwendung erreicht werden kann.

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. Zur abschließenden Analyse und Bewertung der CI/CD-Tools wird das AHP-Verfahren angewandt. Dabei sollen die in den Gesprächen erhobenen Daten als Entscheidungsgrundlage zur Durchführung des Analyseverfahrens verwendet werden.

3.1 Semistrukturierte Leitfadeninterviews zur Erhebung qualitativer Daten

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, S. 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, S. 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 Bestimmung des Gesprächsthemas, 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, S. 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 auftkommende 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, S. 244 ff.]. 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 i.d.R. eine deduktive bzw. induktive Methode verwendet. Bei einer deduktiven Auswertung werden Aussagen der Experten vordefinierten Kategorien zugeordnet [8, S. 244 ff.]. Dieses Evaluationsverfahren bietet insbesondere zur Validierung einer vorabdefinierten Forschungshypothese einen erheblichen Mehrwert. Da im Rahmen dieser Arbeit stattdessen die Beantwortung einer offenen Fragestellung vorgesehen ist, wird eine induktive Kodierung der Interviews vorgenommen. Statt Kategorien im Voraus festzulegen, werden diese bei der induktiven Kodierung dynamisch aus dem Interviewmaterial abgeleitet. Eine implizite Auswertung der induktiven Kodierung wird im AHP-Verfahren (s. Kap. 4) angewendet. So werden die während der Evaluation getroffenen Entscheidungen anhand der Expertenaussagen referenziert.

3.2 Evaluation von Integrations- und Bereitstellungs-Tools 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. [14, 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 beliebig durch einen hierarchischen Aufbau gestaltet werden (s. Abb. 1).

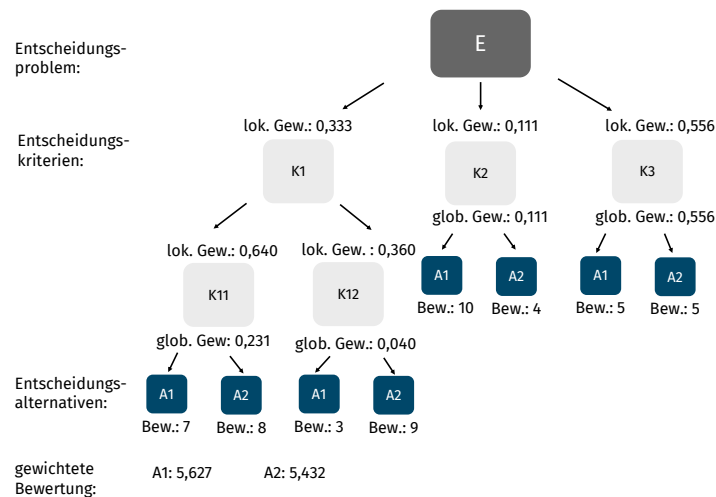


Abbildung 10: Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP. Eigene Darstellung.

Auf oberster Ebene des AHP-Baums befindet sich das Entscheidungsproblem. Es 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 ist für das AHP-Verfahren nach Saaty ein paarweiser Vergleich vorhergesehen. 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 [14, S. 86]. Um den Prozess der Entscheidungsfindung für die Probanden intuitiver zu gestalten, wird von der in der Literatur etablierten Vorgehensweise abgewichen und eine eigene Methode konzipiert. So 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.

Vergleichskriterien (K _i)						
Basiskriterien (K _i)	Kriterium K1	Kriterium K2	Kriterium K3	V _{Ki}	lok. Gew (W _{Ki})	
	Kriterium K1	1	2	0	3	0,333
	Kriterium K2	0	1	0	1	0,111
	Kriterium K3	2	2	1	1	0,556

Tabelle 1: Exemplarische Darstellung der Paarvergleichsmatrix im AHP.
Eigene Darstellung.

So wird dem in Abb. 1 exemplarisch dargestellten Kriterium K1 eine wesentlichere Bedeutung als K2 zugeschrieben. Das korrespondierende Äquivalent auf der rechten Matrixhälfte besitzt entsprechend eine Gewichtung von null (*weniger wichtig*). Um die Zuverlässigkeit der Paarvergleichsgewichtungen zu gewährleisten ist, für das AHP in der Literatur die Berechnung eines Konsistenzindex vorgesehen. Angesichts des hohen Zeitaufwands, welchen die Überprüfung transitiver Beziehungen bei der Durchführung der Gewichtung durch die Experten erfordern würde, wurde auf eine derartige Analyse im Rahmen dieser Arbeit verzichtet. 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:

$$V_{ki} = X_{KiKj+1} + X_{KiKj+2} + X_{KiKj+n}$$

Anschließend muss diese Summe standardisiert werden. Hierfür wird die Gewichtungssumme (V_{ki}) eines Kriteriums durch die Gesamtanzahl der in einer Paarvergleichsmatrix (N) vergebenen Punkte dividiert. Diese ergibt sich durch die Anzahl der Kriterien (n) einer Paarvergleichsmatrix:

$$N = n \cdot n$$

Anschließend kann die prozentuale Gewichtung eines Entscheidungskriteriums wie folgt berechnet werden:

$$W_{ki} = \frac{V_{ki}}{N}$$

Diese lokale Gewichtung wird für jede Hierarchieebene durchgeführt. Um die globale Gewichtung (P_{ki}) zu ermitteln, wird die Gewichtung jedes Subkriteriums mit den Gewichtungen der übergeordneten Kriterien multipliziert:

$$P_{ki} = p_1 \cdot p_2 \cdot p_3 \cdot p_n$$

In der Literatur wird vorgesehen, dass auf unterster Hierarchieebene ebenfalls eine Gewichtung der Entscheidungsalternativen vorgenommen wird, um eine Bewertung durchzuführen [14, 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. 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 optimales Modell. Das Rahmenwerk eignet sich aufgrund des multidimensionalen Entscheidungsmodells besonders für die in der

Arbeit zu untersuchenden 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 (s. Kap. 4.3). Dies ermöglicht eine auf die Präferenzen der Entscheidungsträger abgestimmte Bewertung der zu untersuchenden Pipelines. Das AHP-Verfahren besitzt darüber hinaus ebenfalls einen Vorteil bezüglich des Bewertungsdesigns. Während bei Methoden wie der SWOT-Analyse ausschließlich qualitative Aspekte berücksichtigt werden, ermöglicht das AHP-Model ebenfalls eine Evaluation quantitativer Bewertungsmetriken. Infolgedessen kann die Definition der Bewertungsmetrik flexibel und in Abhängigkeit des Entscheidungskriteriums getroffen werden. Im Rahmen dieser Arbeit wird anhand des AHP-Verfahrens ein für CEA optimales CI/CD-Tool bestimmt. Dabei ist jedoch zu beachten, dass bei der Wahl einer CI/CD-Pipeline ebenfalls „K.O.-Kriterien“ existieren können, welche dazu führen, dass das Rahmenwerk für die Entscheidung keine relevante Aussagekraft mehr besitzt. Deshalb werden bei der Entwicklung der Handlungsempfehlung (s. Kap. 5) Szenarien definiert, bei welchen die Wahl der CI/CD-Pipeline vom AHP-Ergebnis abweicht. Weitere Erläuterungen zu den im AHP-Verfahren getroffenen Entscheidungen werden zur besseren Verständlichkeit in Kapitel 4 ausgeführt.

4 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*. Bei der Untersuchung der Pipelines sind dabei folgende Rahmenbedingungen zu beachten: Im Rahmen der Arbeit soll evaluiert werden, ob sich die verschiedenen Pipelines zur Bereitstellung von Software für CEs eignen. Weiterhin muss festgestellt werden, inwiefern die Tools für die Technologien SAP CAP bzw. SAP UI5 kompatibel sind und ob eine Bereitstellung in der Laufzeitumgebung Cloud Foundry der SAP BTP möglich ist.

4.1 Definition der Entscheidungskriterien

Die zur Durchführung des AHP-Verfahrens benötigten Daten werden neben einer Literaturrecherche ebenfalls mittels Experteninterviews erhoben. Für die Experteninterviews wird ein Gremium aus acht Mitarbeitenden der SAP zusammengestellt (s. Anhang C). Diese sind jeweils in verschiedenen Bereichen der Cloud-Fullstack-Entwicklung, Test-Management, Product Management sowie in der Softwarearchitektur spezialisiert. Somit kann Expertise über verschiedene Fachbereiche hinweg aufgebaut und Anforderungen aller an der Entwicklung, Bereitstellung sowie an dem Betrieb von Software beteiligten Stakeholdern erfasst werden. Zur Festlegung der Entscheidungskriterien wird eine induktive Kodierung der Expertengespräche durchgeführt (s. Anhang C.5). Dabei werden aus besonders häufig von Experten genannten Aspekten systematisch Kategorien abgeleitet. Diese umfassen insbesondere Aspekte, welche die in vergangenen Kundenprojekten hervorgegangenen Anforderungen an eine CI/CD-Pipeline widerspiegeln. 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:










	K1 Funktionalität
	K1.1 Tests K1.2 Code-Analysen K1.3 Build K1.4 Deploy K1.5 Monitoring
	K2 Integrationsmöglichkeiten
	K2.1 Integration von Repositories K2.2 Integration von Entwicklungsumgebungen K2.3 Integration von Planungs-Software
	K3 Kosten
	K4 Skalierbarkeit
	K5 Performance
	K5.1 Integration-Zeit K5.2 Delivery-Zeit
	K6 Flexibilität
	K7 Support
	K7.1 Administrativer Support K7.2 Community Support
	K8 Sicherheit
	K8.1 Authentifizierung und Autorisierung K8.2 Sicherheitsarchitektur
	K9 Benutzerfreundlichkeit
	K9.1 Installation und Wartung K9.2 Intuitive Bedienbarkeit

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 laut Experte 1 etwa dazu in der Lage sein, Anwendungen zu testen, Code-Analysen durchzuführen und eine Software auf der Cloud-Plattform bereitzustellen [0, Z. 72 ff.]. Angesichts der Vielfältigkeit des Entscheidungskriteriums Funktionalität wird eine Untergliederung in verschiedene Subkriterien vorgenommen. In Kriterium K1.1 wird dabei die Unterstützung automatisierter

Unit-Tests evaluiert [0, Z. 72 ff.]. 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 Funktionalitäten kontinuierliche und automatisierte durchgeführte Prüfung [64, Z. 65 ff.]. Dies umfasst eine Abwicklung von Unit-, Integration-, E2E-Test sowie Regression-Tests für Backend sowie Frontend. Hinsichtlich der Entwicklung von CAP-Node-Anwendungen wird in der SAP die Durchführung von Unit-Tests mittels Jest bzw. Mocha und Integration-Tests mittels Newman vorgeschlagen. Für die Programmierung mit SAP UI5 sind hingegen Unit-Tests mittels Q-Unit, Integration-Tests mittels OPA5 und E2E-Tests mittels WDI5 vorgesehen [64, Z. 66 ff.]. Für Regression-Tests wird hingegen kein spezifisches Test-Framework verwendet. In diesem Zusammenhang wird lediglich überprüft, ob eine erneute Validierung bereits bestehender Softwarekomponenten durchführbar ist. In Kriterium K1.2 wird die Kompatibilität verschiedener *Code-Analyse-Tools* untersucht [0, Z. 72 ff.]. Es wurde gezielt eine Trennung dieser Kategorie mit dem Kriterium K1.2 (Tests) vorgenommen. Während Tests eine funktionale Erfüllung der Anforderung evaluieren, werden mit den Analysen Code-Qualitätsstandards der entwickelten Funktionalitäten untersucht. Dabei wird evaluiert, ob statische Codeanalysen, Security- sowie Performance-Überprüfungen von den CI/CD-Pipelines unterstützt werden. Laut Experte 3 werden für statische Code-Analysen gemäß Produktstandards der SAP Lint sowie SonarQube verwendet [69, Z. 44 ff.]. Zur Durchführung von Sicherheitsüberprüfungen wird für SAP-CAP-Node-Anwendungen Checkmarx verwendet, während für SAP UI5 DASTER zum Einsatz kommt. Für Performance-Tests wird das Tool JMeter verwendet. In der Kategorie K1.3 werden die *Build-Funktionalitäten* der CI/CD-Pipelines evaluiert [0, Z. 71 ff.]. Dabei wird untersucht, ob die Entscheidungsalternativen das Build-Tool Make zum Kompilieren von Multi-Target-Applications (MTAs) unterstützen. 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. MTAs werden dabei i.d.R. für die

Bereitstellung von SAP-CAP-Node- sowie SAP-UI5-Anwendungen verwendet [44]. Laut Experte 1 ist weiterhin essenziell, dass Pipelines Artefakt-Repositories unterstützen [0, Z. 15 ff.]. Eine MTA kann von verschiedenen externen Ressourcen (z.B. Bibliotheken) abhängig sein. Diese sind in einem Artefakt-Repository verwaltete externe Komponenten, welche bei der Entwicklung neuer Microservices wiederverwendet werden können [0, Z. 40 ff.]. Diese Komponenten werden während des Build-Prozesses von der CI/CD-Pipeline aus den Artefakt-Repositories geladen und kompiliert. Ein weiterer für CEs essenzieller Build-Aspekt ist die Unterstützung von Docker-Workflows. 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 [21]. In Kriterium K1.4 werden die *Deploy- und Release-Funktionalitäten* der CI/CD-Tools untersucht [0, Z. 73 ff.]. Dabei wird evaluiert, ob die Pipelines neben dem Ausrollen von Software auf die Cloud-Foundry-Laufzeitumgebung ebenfalls verschiedene Bereitstellungsstrategien unterstützen (z.B. Blue/Green-Deployment s. Kap. 2.2.3). Des Weiteren wird erörtert, ob mit den CI/CD-Pipelines eine Bereitstellung in das SAP Cloud Transport Management (SAP CTM) möglich ist. Das SAP CTM kann als zusätzliche Schicht im CI/CD-Prozess verbaut werden (s. Abb. 15). Experte 2 merkt an, dass dieses System dabei insbesondere innerhalb komplexer ERP-Landschaften zur Optimierung der Bereitstellungsprozesse führen kann (weitere Details in Kap. 4.4)[68, Z. 65 ff.]. In Kategorie K1.5 wird die *Monitoring-Funktionalität* der verschiedenen CI/CD-Pipelines untersucht [68, Z. 37 ff.]. In diesem Zusammenhang erfolgt eine Bewertung der Überwachbarkeit der CI/CD-Tools. Für interne Projekte wird dabei i.d.R. das SAP-Partner-Tool Splunk verwendet [68, Z. 74 ff.]. Damit lassen sich verschiedene Metriken, wie Build-Zeiten, Performance-Checks oder Fehlerquoten verschiedener Pipelines in einem zentralisierten Dashboard visualisieren. Da CI/CD-Pipelines im Rahmen dieser Arbeit ebenfalls für externe Kundenprojekte evaluiert werden, ist innerhalb dieses Kriteriums ebenfalls eine Betrachtung von externen Open-Source-Tools vorgesehen. Laut Experte 2 wird dabei häufig das Kibana-Dashboard verwendet [68, Z. 74 ff.]. In Katego-

rie K2 werden die **Integrationsmöglichkeiten** der Pipelines untersucht. In dem Subkriterium *Integrationsmöglichkeiten von Repositories (Kategorie K2.1)* wird evaluiert, ob sich das Repository in die Pipeline integrieren lässt [0, Z. 89 ff.]. Damit können bestimmte Ereignisse, wie Push-Mitteilungen bei Code-Änderungen automatisiert an die CI/CD-Pipeline übermittelt werden. Somit kann ein unmittelbarer Integrations- bzw. Bereitstellungs-Workflow ausgelöst werden. Bei der Bewertung wird ein besonderes Augenmerk darauf gelegt, dass häufig verwendete Repositories problemlos in die Pipeline integrierbar sind. Da in der Literatur diesbezüglich keine öffentlichen Statistiken zugänglich sind, wird auf empirische Einschätzungen der Experten zurückgegriffen. So wurden nach Beurteilung des Experten 3 innerhalb interner und externer Projekte am häufigsten GitHub, GitLab und BitBucket verwendet [65, Z. 95 ff.]. In Kriterium K2.2 werden die *Integrationsmöglichkeiten von Entwicklungsumgebungen* untersucht [0, Z. 93 ff.]. Die Integration-Pipeline kann unmittelbar während des Entwicklungsprozesses aus der Entwicklungsumgebung gestartet werden. Auf diese Weise wird sichergestellt, dass Entwickler Feedback in noch kürzeren Zeitabständen erhalten als bei einer ausschließlichen Integration der Pipeline in das Repository. Die Bewertung bezieht sich dabei ausschließlich auf SAP-UI5- sowie SAP-CAP-Node-Entwicklungsumgebungen. Dazu gehören Microsoft Visual Studio Code, SAP Business Application Studio (SAP BAS) sowie Eclipse. In Kriterium K2.3 wird die *Integrationsmöglichkeit von Planungssoftware* untersucht [65, Z. 96 ff.]. Dazu gehören Projektmanagement-Tools wie Jira. Laut Experte 4 ermöglicht die Integration einer solchen Planungssoftware, Projektmanager eine erhöhte Transparenz über den Bereitstellungs-Workflow aller zu implementierender Arbeitselemente zu erhalten [65, Z. 96 ff.]. Auf diese Weise kann der CI/CD-Status eines Backlog-Items unmittelbar über die Planungssoftware eingesehen werden. Da die Wahl eines Pipeline-Tools i.d.R. nicht von der Unterstützung eines bestimmten Projektmanagement-Tools abhängig gemacht wird, erfolgt lediglich eine Untersuchung der generellen Integrationsfähigkeit von Planungssoftware [65, Z. 96 ff.]. In Kriterium K3 erfolgt die Evaluation der **Kosten** [68, Z. 42 ff.]. Mit diesem Entscheidungskriterium werden die durch die CI/CD-Pipelines verursachten *Total Cost*

of Owner Ship (TCO) evaluiert. TCO beschreiben den Geldbetrag, welchen ein Unternehmen während des gesamten Lebenszykluses einer Investition, also von Beschaffung bis zur vollständigen Entsorgung, aufbringen muss [3, S. 3]. Da für die zu untersuchenden Pipeline-Tools keine vergleichbaren Kostenmodelle verfügbar sind, werden die Kosten ausschließlich qualitativ beurteilt. Somit werden Aspekte wie Einrichtungs-, Betriebs- sowie Wartungsaufwendungen evaluiert. In Kriterium K4 wird die **Skalierbarkeit** der CI/CD-Pipelines analysiert [0, Z. 69 ff.]. Hierbei werden die Pipelines auf horizontale sowie 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. Laut Experte 1 kann die CI/CD-Pipeline so dynamisch an die sich ändernden Anforderungen angepasst werden [0, Z. 74 ff.]. In Kriterium K5 wird die **Performance** der Pipelines verglichen [68, Z. 35 ff.]. Dabei wird die zur Prozessierung des CI/CD-Workflows benötigte Zeit der zu untersuchenden Tools anhand eines für die Arbeit implementierten CEA-Prototyps evaluiert (s. 17). 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-Branche bis zur vollständigen Konsolidierung in den Hauptzweig benötigt wird. Dabei werden für die Microservices Validierungen, wie Unit- sowie Integration-Tests, welche für gewöhnlich in einem CI-Prozess abgewickelt werden, implementiert und in die Pipeline eingebunden. Die Delivery-Zeit beschreibt die Zeitspanne, welche von der Freigabe des Hauptzweigs bis zur Bereitstellung der Software auf die Cloud-Plattform benötigt wird. Dabei werden ebenfalls CD-typische Schritte, wie E2E-Tests und Code-Analysen implementiert und in die Pipelines integriert. In Kriterium K6 wird die **Flexibilität** der verschiedenen Pipelines evaluiert [0, Z. 70 ff.]. Eine bedeutende Dimension der Flexibilität ist die uneingeschränkte Konfigurierbarkeit der Pipelines. So sollte eine Pipeline laut Experte 1 etwa keinerlei Beschränkungen in Bezug auf Anzahl und Reihenfolge der im CI/CD-Workflow durchzuführenden Schritte besitzen [0, Z. 70 ff.]. Weiterhin

wird evaluiert, ob die Pipelines einen modularen Aufbau unterstützen. Um die aus einer Bereitstellungslandschaft mit einer Vielzahl an CI/CD-Pipelines resultierende Komplexität zu reduzieren, sollten Pipelines aus modularen wiederverwendbaren Komponenten zusammengesetzt werden können. Sofern eine neue Pipeline erforderlich ist, besteht die Möglichkeit diese ohne hohen Implementierungsaufwand aus den wiederverwendbaren CI/CD-Komponenten zu konstruieren. Ein weiterer, für die Flexibilität der Pipelines essenzieller Aspekt ist die Unterstützung von Plug-ins. Damit diesen ebenfalls nicht im Standard verfügbare Funktionalitäten in die Pipeline integriert werden können, besteht für die Entwicklungsabteilung die Möglichkeit flexibel auf Anforderungen aller Stakeholder zu reagieren. In Kriterium K7 wird der für die CI/CD-Pipelines bereitgestellte **Support** evaluiert. Im Hinblick auf den *Administrativen Support (K7.1)* wird geprüft, ob die Pipeline-Anbieter Unterstützung bei der Einrichtung, Konfiguration sowie Problembehebung der CI/CD-Tools bieten [68, Z. 44 ff.]. Dies ist insbesondere dann hilfreich, wenn der Umgang mit den Pipelines einen hohen Grad an Expertise erfordert. 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 neuesten Stand der Technik ist. Im Kontext des *Community-Supports (Kriterium K7.2)* wird geprüft, ob öffentliche Foren existieren, in welchen Anwender Fragen stellen und Probleme diskutieren können [68, Z. 46 ff.]. Die Qualität des Community-Supports hängt dabei i.d.R. von dem Kontributionsmaß innerhalb der Foren ab. Somit wird als Bewertungsgrundlage die Anzahl der zu einer CI/CD-Pipeline abgesetzten Posts verglichen. Als Referenzquelle dient hierbei das größte Entwicklerforum Stack Overflow [54]. In dem Kriterium K8 wird die **Sicherheit** der CI/CD-Pipelines untersucht [0, Z. 79 ff.]. Um unerwünschte Zugriffe zu vermeiden, sollte die CI/CD-Pipeline ein Authentifizierungs- und Authentisierungskonzepte unterstützen. Besonders vorteilhaft ist dabei die Einbindung zentralisierter Drittanbieter, wie der SAP Identity Provider oder GitHub. Ein weiteres unter dem Kriterium der Sicherheit evaluierter Aspekt ist ebenfalls die Systemsicherheit. Dazu gehört neben dem Schutz der Systemintegrität ebenfalls die

Ausfallsicherheit. In Kriterium K9 wird die **Benutzerfreundlichkeit** der CI/CD-Pipelines untersucht. Hinsichtlich der *Installation und Wartung (Kriterium K9.1)* ist es dabei besonders vorteilhaft, wenn Installation und Wartung des CI/CD-Tools nicht selbst übernommen werden muss, sondern unmittelbar als Service bereitgestellt wird [0, Z. 67 ff.]. Auch der für die Implementierung und Konfiguration der Pipelines benötigte Aufwand sollte so gering wie möglich sein (*intuitive Bedienbarkeit*) [0, Z. 67 ff.]. Um die Abhängigkeit einer Abteilung von hochqualifizierten DevOps-Spezialisten zu verringern, kann es einen Mehrwert darstellen, wenn Pipelines nicht mittels Programmiersprachen, sondern über intuitive Benutzeroberflächen konfigurierbar sind.

4.2 Festlegung der Bewertungsmetriken

In diesem Abschnitt erfolgt die Festlegung der Bewertungsmetriken. Dabei ist eine Bewertung von null bis vier Punkte vorgesehen. Eine Bewertung mit vier Punkten wird vergeben, wenn eine CI/CD-Pipeline signifikant zur Zielerreichung eines Kriteriums beiträgt, während eine Bemessung von null Punkten eine unterdurchschnittliche Leistung impliziert. Die Vergabe von null Punkten stellt sicher, dass ein Kriterium, falls die zu bewertende CI/CD-Pipeline keinen Mehrwert birgt, nicht zur Erhöhung der Gesamtbewertung beiträgt.

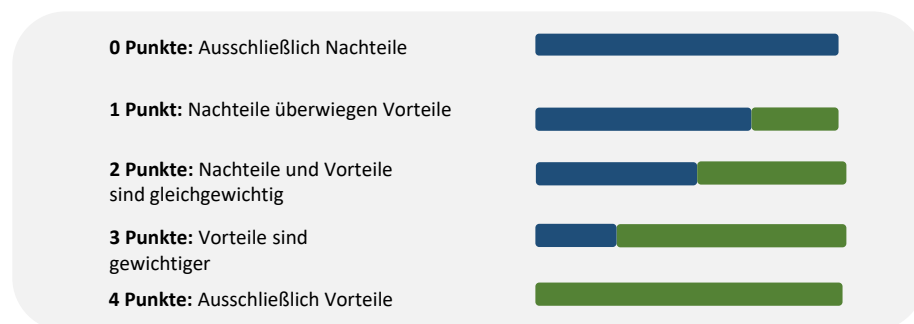


Abbildung 12: Qualitative Bewertungsmetrik für AHP. Eigene Darstellung.

Für qualitative, in dem Entscheidungs-Framework zu betrachtende Kriterien wird eine gewichtende Bewertung vorgenommen. Dabei erfolgt eine Abwägung der Vor- und Nachteile, welche sich aus der Nutzung einer bestimmten Pipeline ergeben. Auf

diese Weise kann bei der Bewertung ebenfalls argumentativ auf die in einer CEA vorliegenden Bedürfnisse eingegangen werden. Aufgrund des qualitativen Evaluations-Designs wird diese Metrik für *Funktionalität (K1)*, *Integrationsmöglichkeiten (K2)*, *Skalierbarkeit (K4)*, *Flexibilität (K5)*, *Administrativer Support (K7.1)*, *Sicherheit (K8)* und *Benutzerfreundlichkeit (K9)* angewendet. Da für die verschiedenen Pipelines unterschiedliche Preismetriken festgelegt sind, kann für das Kriterium *Kosten (K3)* ebenfalls ausschließlich eine qualitative Erörterung abgewickelt werden. Für das quantitative Kriterium *Performance (K5)* wird eine divergierende Metrik definiert:

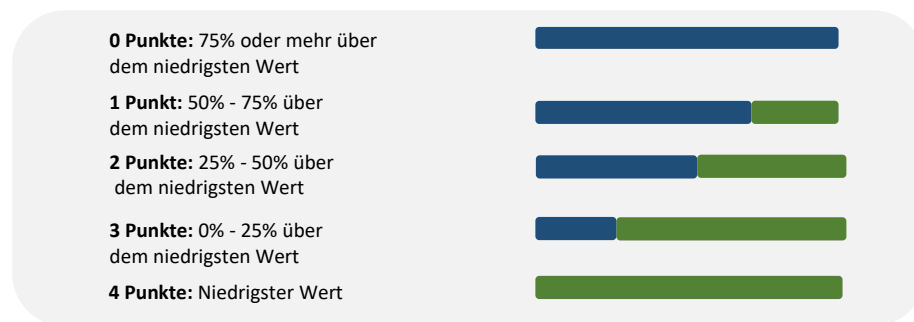


Abbildung 13: Quantitative Bewertungsmetrik für AHP. Eigene Darstellung.

Mithilfe dieser Metrik lassen sich die relativen Kosten der CI/CD-Pipelines vergleichend betrachten. Indem der höchste Wert als Bezugsgröße verwendet wird, ermöglicht sich ein Vergleich in Relation zur besten Entscheidungsalternative. Das vorliegende Evaluations-Design erweist sich dabei als besonders vorteilhaft, da dieses eine Bewertung ermöglicht, ohne vorab spezifische Referenzwerte festzulegen. In Kriterium K7.2 (*Community-Support*) ist ebenfalls eine quantitative Bewertung vorgesehen. Dabei wird die Anzahl der abgesetzten Blog-Posts zu einer CI/CD-Lösung evaluiert. Hierbei erfolgt eine inverse Bewertung der in Abb. 13 dargestellten Metrik. So werden vier Punkte für die höchste Blog-Post-Anzahl vergeben. Für die restlichen Bewertungen wird invers nach dem in Abb. 13 definierten Abstufungen benotet.

4.3 Ermittlung der Gewichtungsfaktoren

Zur Bestimmung einer optimalen, auf die Präferenzen der Entscheidungsträger ausgerichteten CI/CD-Pipeline wird eine Gewichtung der Bewertungskriterien vorgenommen. Die Gewichtung wird dabei von einem Expertengremium, bestehend aus fünf Mitarbeitenden der SAP, durchgeführt. Dieses umfasst einen Softwarearchitekten, einen Backend- sowie Frontend-Test-Entwickler, einen Fullstack-Entwickler und einen Product Manager (s. Anhang C.6). Dabei wird die in Kapitel 3.2 erläuterte Paarvergleichgewichtung von jedem Experten zunächst eigenständig durchgeführt. Im Anschluss erfolgt die Berechnung des Mittelwertes aller Einzelgewichtungen. Mit diesem Vorgehen wird gewährleistet, dass bei der Gewichtung divergierende Anforderungen und Bedürfnisse aller an dem Bereitstellungsprozess von Software beteiligten Stakeholder adäquat berücksichtigt werden.










Kriterium	Lokale Gewichtung pro hundert	Globale Gewichtung pro hundert
 K1 Funktionalität	0,1728	0,1728
K1.1 Tests	0,3200	0,0548
K1.2 Code-Analysen	0,2080	0,0212
K1.3 Build	0,2240	0,0380
K1.4 Deploy	0,1200	0,0390
K1.5 Monitoring	0,1280	0,0198
 K2 Integrationsmöglichkeiten	0,1580	0,1580
K2.1 Integration in Repository	0,2889	0,0796
K2.2 Integration in Entwicklungsumgebung	0,5111	0,0444
K2.3 Integration in Planungs-Software	0,2000	0,0340
 K3 Kosten	0,0963	0,0963
 K4 Skalierbarkeit	0,1160	0,1160
 K5 Performance	0,0790	0,0790
K5.1 Integration-Zeit	0,7000	0,0574
K5.2 Delivery-Zeit	0,3000	0,0216
 K6 Flexibilität	0,1185	0,1185
 K7 Support	0,0815	0,0815
K7.1 Administrativer Support	0,3500	0,0290
K7.2 Community Support	0,6500	0,0525
 K8 Sicherheit	0,1259	0,1259
 K9 Benutzerfreundlichkeit	0,0519	0,0519
K9.1 Installation und Wartung	0,4000	0,0222
K9.2 Intuitive Bedienbarkeit	0,6000	0,0296

Abbildung 14: Gewichtungsfaktoren der AHP-Entscheidungskriterien. Eigene Darstellung.

In Abb. 14 werden die lokalen und globalen Durchschnittsgewichtungen der AHP-Entscheidungskriterien dargestellt (s. Abb. 14). Die Experten legen dabei auf unterschiedliche Aspekte Wert. So ist für den Softwarearchitekten die von den Pipelines

bereitgestellte Funktionalität von hoher Bedeutung. Für den Experten besonderes ausschlaggebend ist dabei die Unterstützung verschiedener Test-Frameworks. Nach seiner Auffassung besteht die Möglichkeit, Tests manuell und unabhängig von einer Pipeline auszuführen. Allerdings gestaltet es sich dabei für Softwarearchitekten sehr herausfordernd, die Einhaltung dieser Testrichtlinien durch die Entwickler angemessen zu überblicken [70, Z. 18 ff.]. Weiterhin ist für den Experten essenziell, dass Pipelines unterschiedliche Build-Tools unterstützen. So ist es bei einer CEA üblich, dass eine hohe Bandbreite verschiedener Programmier-Frameworks eingesetzt wird [70, Z. 13 ff.]. Da Pipeline-Systeme i.d.R. einmalig aufgesetzt werden, ist die Installation und Wartung für den Softwarearchitekten weniger wichtig. Sowohl für den Fullstack- als auch die Test-Entwickler stellt die Unterstützung automatisierter Tests einen signifikanten Aspekt dar [63, Z. 5 ff.]. Da die Entwickler ein möglichst zeitnahes Feedback auf Erweiterungen erhalten möchten, ist dabei insbesondere die Integration-Zeit essenziell [64, Z. 23 ff.]. Kosten stellen für die Entwickler hingegen eine geringe Relevanz dar, da diese während der operativen Tätigkeit kaum Berührungspunkte mit diesem Aspekt aufweisen. Eine höhere Bedeutung besitzt für die Entwickler hingegen der Community-Support [66, Z. 25 ff.]. Dieser kann als Hilfestellung zur Implementation der Pipelines verwendet werden. Für den Product Manager sind die Integrationsmöglichkeiten ein wichtiger Aspekt. Gemäß empirischer Erkenntnisse ist die Wahl eines CI/CD-Tools, insbesondere von der Kompatibilität mit dem Repository abhängig [67, Z. 5 ff.]. Weiterhin erachtet der Experte Sicherheit als einen essenziellen Aspekt. Da die Bereitstellung von Software zur Wertschöpfung beiträgt, können Unterbrechungen der CI/CD-Pipelines erhebliche Auswirkungen auf die Wettbewerbsfähigkeit besitzen. Überdies bieten CI/CD-Pipelines eine effektive Möglichkeit, um Schadsoftware in die Produktionssysteme einzuschleusen [67, Z. 5 ff.].

4.4 Bewertung der Entscheidungsalternativen

Das Erste zu bewertende Kriterium ist die **Funktionalität**. Sowohl für Jenkins als auch für Azure Pipelines wird die Programmbibliothek Project Piper verwendet. Mit

dieser werden essenzielle, für die Bereitstellung von SAP-Technologien verwendete Pipeline-Funktionalitäten ausgeliefert. Daher erzielen beide CI/CD-Tools innerhalb der *Funktionalität* weitgehend ähnliche Ergebnisse. In Kriterium K1 (*Tests*) wird die Unterstützung von Unit-, Integration-, E2E-, sowie Regression-Tests evaluiert. Mit Project Piper wird eine Test-Laufzeitumgebungen für Node zur Verfügung gestellt. Somit ist es möglich, Backend-Unit-Tests mittels Mocha und Jest auszuführen. Die Programmbibliothek unterstützt ebenfalls die für Frontend-Unit-Tests mittels Qunit und Frontend-Integration-Tests mittels OPA5 benötigte Laufzeitumgebung Karma. Darüber hinaus wird das für Backend-Integration-Tests benötigte Newmann-Tool bereitgestellt. Für E2E-Tests mittels WDI5 wird eine Webdriver-Laufzeitumgebung ausgeliefert [65, Z. 66 ff.]. Im Kontext der CEA stellt ebenfalls die Automatisierung von Regression-Tests einen essenziellen Faktor dar. Bei der Weiterentwicklung eines Microservice muss validiert werden, ob abhängige Dienste stets ordnungsgemäß funktionieren. Dafür wird mit Azure Pipelines und Jenkins ein Ressourcen-Trigger bereitgestellt [32]. Dieser gewährleistet ein automatisiertes Ausführen von Pipelines abhängiger Microservices. Dabei erfolgt die Veröffentlichung des neuen Artefakts erst, nachdem alle Tests validiert wurden. In SAP CI/CD können alle Tests, mit Ausnahme der Backend-Integration-Tests mittels Newmann und Regression-Tests ausgeführt werden. Dies stellt insbesondere für CEs einen erheblichen Nachteil dar. Diese sind aufgrund ihrer modularen IT-Architektur darauf angewiesen, dass einzelne Microservices reibungslos miteinander interagieren. Da ein implizites Validieren des Komponentenzusammenspiels ebenfalls über E2E-Tests abgewickelt wird, fällt dieser Nachteil jedoch weniger gewichtig aus. Aus diesem Grund wird eine Bewertung von drei Punkten für SAP CI/CD vergeben (Vorteile überwiegen Nachteilen). Für Azure Pipelines sowie Jenkins ist eine Bewertung von vier Punkten vorgesehen (ausschließlich Vorteile). In Kriterium K1.2 erfolgt eine Validierung der durch die Pipeline bereitgestellten *Code-Analyse-Funktionalitäten*. Project Piper unterstützt statische Code-Analyse mittels Lint bzw. SonarQube, Sicherheitsüberprüfungen mittels Checkmarx und DASTER sowie Performance-Tests mittels JMeter [69, Z. 40 ff.]. Mit dem SAP CI/CD-Tool sind ausschließlich statische Code-Analysen mittels

Lint bzw. SonarQube möglich [0, Z. 50 ff.]. Neben der fehlenden Unterstützung von Performance-Tests birgt insbesondere die Inkompatibilität von Sicherheits-Tools für CEA erhebliche Nachteile. Die CE-typische Verwendung von APIs zur Kommunikation zwischen einzelnen Microservices hat zur Folge, dass eine für unautorisierte Zugriffe begünstigte Angriffsfläche entsteht. Obwohl mögliche Sicherheitsbedenken auch durch Security-Experten manuell behoben werden könnten, ist dies für die von CEs angestrebte dynamische Reaktionsfähigkeit hinderlich. Zudem wird die Abwicklung von *Static Application Security Testing (SAST)* mit Checkmarx bzw. *Dynamic Application Security Testing (DAST)* mit DASTER von der SAP als Produktqualitätsstandard vorgeschrieben [69, Z. 37 ff.]. Somit kann die SAP CI/CD-Pipeline nicht von Kunden verwendet werden, welche sich an dem Sicherheitsstandard der SAP orientieren. Aus diesem Grund ergibt sich eine Bewertung von einem Punkt für den SAP CI/CD-Service (Nachteile überwiegen) und vier Punkten für Jenkins bzw. Azure Pipelines (ausschließlich Vorteile). In Kriterium K1.3 wird die *Build-Funktionalität* der Pipelines evaluiert. Mit Project Piper wird dabei das für SAP UI5 sowie SAP CAP Node benötigte MTA-Build-Tool unterstützt. Weiterhin wird durch die Programmbibliothek ein Build-Tool für Docker-Container bereitgestellt [28]. Die kompilierten Anwendungsprogramme können mit Project Piper in einem Artefakt-Repository bereitgestellt werden. SAP CI/CD unterstützt kein Docker-Workflow sowie Artefakt-Repository [50]. Besonders gewichtig ist dabei die fehlende Unterstützung des Artefakt-Repositories. Artefakt-Repositories spielen für CEs eine essenzielle Rolle bei der Durchführung von Rollbacks. Bei dem Auftreten von Fehlern kann zu einer im Artefakt-Repository abgelegten früheren Version zurückgekehrt werden. Somit wird das Risiko von Ausfällen und Unterbrechungen im Geschäftsbetrieb minimiert. Da der SAP CI/CD-Service dennoch das für SAP CAP Node und SAP UI5 benötigte MTA-Build-Tool bereitstellt und somit den minimal erforderlichen Satz an benötigter Build-Funktionalität unterstützt, wird eine Bewertung von zwei Punkten veranschlagt (Nachteile und Vorteile gleichgewichtig). Für Azure Pipelines sowie Jenkins wird eine Bewertung von 4 Punkten vergeben (ausschließlich Vorteile). In Kriterium K1.4 erfolgt die Evaluierung der *Deploy- und*

Release-Funktionalitäten. Dabei herrscht bei SAP CI/CD, Azure Pipelines sowie Jenkins Feature-Parität. Ein erheblicher Mehrwert besteht insbesondere darin, dass alle zu untersuchende CI-CD-Lösungen eine Bereitstellung in das SAP CTM unterstützen. Durch den Einsatz dieses Tools lässt sich die Bereitstellung von Software innerhalb komplexer ERP-Systemlandschaften optimieren. Ein Unternehmen könnte etwa separate Systeme für das Entwickeln (*DEV*), Testen (*TEST*) sowie für die Produktionsumgebung (*PROD*) besitzen. Während die Verantwortung der Entwickler dabei auf die ordnungsgemäße Bereitstellung der Funktionalitäten in das DEV-System beschränkt ist, werden nachfolgende Schritte von dem Betriebsteam über das SAP CTM verwaltet. Über dieses System können dabei vielfältige Release-Konfigurationen, wie z.B. eine Zeitplan-gesteuerte Bereitstellung vorgenommen werden. Des Weiteren ermöglicht das SAP CTM die Definition von Abhängigkeiten verschiedener Services. Experte 2 merkt an, dass dies insbesondere für CEA von hoher Bedeutung ist [68, Z. 67 ff.]. Im Falle einer API-Änderung bestimmter Microservices, besteht die Möglichkeit, dass in konsumierenden Diensten entsprechend Fehler auftreten. Mittels des SAP CTMs kann dabei jedoch reguliert werden, dass die neue Version eines Microservices erst nach Anpassung der abhängigen Dienste in das Produktivsystem eingeführt wird (s. Abb. 15).

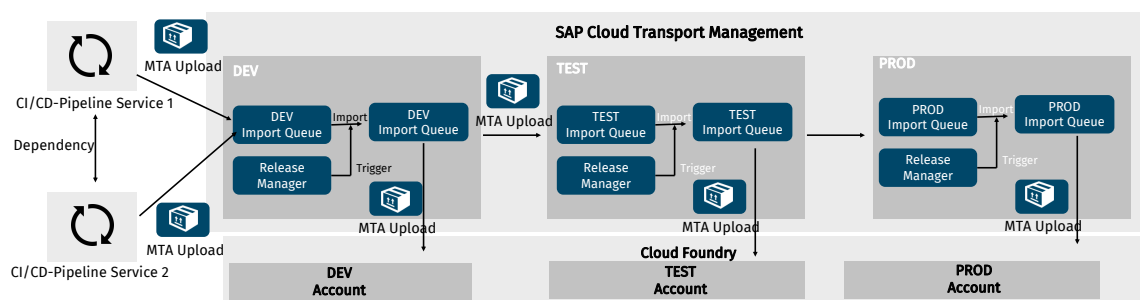


Abbildung 15: SAP Cloud Transportmanagement. In Anlehnung an Stevens [55].

Weiterhin wird von den Pipelines das Ausrollen einer neuen Anwendungsversion mittels Blue/Green-Deployment unterstützt [30]. Diese Strategie gewährleistet die

notwendige Flexibilität und Agilität, welche es in CEs benötigt. Da im Blue-Green-Deployment neben der zu installierenden neuen Version ebenfalls die stabile Anwendung betrieben wird, können mit dieser Bereitstellungsstrategie Unterbrechungszeiten vermieden werden. Dies spielt insbesondere im Kontext von Composable-ERP-Systemen, in welchem etwa kritische Finanzprozesse abgewickelt werden, eine bedeutende Rolle. Während alle CI/CD-Tools eine Cloud-Foundry-, CTM- sowie Blue-Green-Bereitstellung unterstützen, ist Canary- sowie Shadow-Deployment nicht möglich. Da diese jedoch ausschließlich ergänzende Zusatzfunktionalitäten und keine essenziell benötigten Werkzeuge darstellen, wird eine Bewertung von 3 Punkten vergeben (Vorteile überwiegen). In Kriterium K1.5 wird die *Monitoring-Funktionalität* der Pipelines untersucht. Mit Project Piper können im CI/CD-Prozess generierte Logs unmittelbar an das Monitoring-Dashboard Splunk übermittelt werden [68, Z. 74 ff.]. Open-Source-Tools wie das Kibana-Dashboard lassen sich dabei ebenfalls mit Jenkins bzw. Azure Pipelines verknüpfen. Während Kibana unmittelbar als Azure-SaaS-Tool verfügbar ist, benötigt das Aufsetzen auf Jenkins einen deutlich höheren Aufwand (s. Abb. 16).

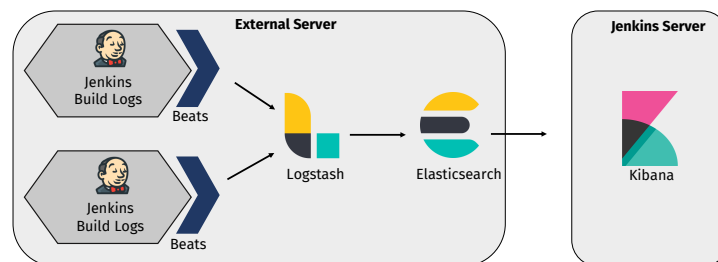


Abbildung 16: Pipeline-Monitoring mit Kibana und Jenkins. In Anlehnung an Atta [22]

So müssen auf der Jenkins-Plattform Tools zum Versenden (Beats), Transformieren (Logstash) bzw. zum Speichern (Elasticsearch) der Pipeline-Logs manuell aufgesetzt werden. Um die Pipeline-Metriken letztlich auf dem Dashboard zu visualisieren, müssen die Logs über APIs an eine extern gehostete Kibana-Instanz übermittelt werden [22]. Der SAP CI/CD-Service unterstützt ausschließlich eine Protokollierung der Build-Logs einzelner Pipelines. Für Experte 2 stellt die Inkompatibilität von

Monitoring-Dashboards insbesondere in einer Microservice-Architektur einen erheblichen Nachteil dar. So sind die zentralen Dashboards bei aus komplexen Diensten bestehenden Systemarchitekturen häufig die einzige Möglichkeit CI/CD-Prozesse nachhaltig zu überwachen [68, Z. 38 ff.]. Während für Azure Pipelines vier Punkte vergeben werden (ausschließlich Vorteile), erhält Jenkins aufgrund des hohen Einrichtungsaufwands für Kibana-Dashboard eine Bewertung von drei Punkten (Vorteile überwiegen). Für die SAP CI/CD-Pipeline wird aufgrund der fehlenden Unterstützung von Monitoring-Dashboards ausschließlich ein Punkt vergeben (Nachteile überwiegen). In Kriterium K2 werden die **Integrationsmöglichkeiten** der Pipelines evaluiert. Alle drei Pipelines unterstützen dabei eine Integration der Repositories GitHub, BitBucket und GitLab (Kriterium K2.1). Für Jenkins und SAP CI/CD müssen dafür jedoch manuell in den entsprechenden Repositories Webhooks aufgesetzt werden. Ein Webhook ist eine API, welche Anfragen zum Auslösen eines CI/CD-Workflows an die Pipeline übermittelt. Diese Webhook-Anfragen können bei einer Vielzahl von Events, einschließlich dem *Pushen* von Codeänderungen oder dem Eröffnen von *Pull-Requests*, ausgelöst werden. SAP CI/CD unterstützt dabei keine Pull-Request-Webhooks. Dies birgt im Entwicklungsprozess erhebliche Nachteile. Eine Pull-Request-Pipeline gewährleistet, dass Entwicklungen eines Feature-Banches erst nach erfolgreicher Abwicklung aller Tests in den Hauptzweig integriert werden [0, Z. 27 ff.]. Ohne diese Funktionalität muss die Pipeline bei Erstellung eines Pull-Requests stets manuell gestartet und überprüft werden, was zur Beeinträchtigung der Zusammenarbeit in Teams führt. Für Azure Pipelines wird eine Bewertung von vier Punkten vergeben (ausschließlich Vorteile). Jenkins erhält aufgrund der Notwendigkeit einer manuellen Webhooks-Konfiguration drei Punkte (Vorteile überwiegen). Da SAP CI/CD darüber hinaus keine Pull-Request-Pipelines unterstützt und diese laut Experte 1 in einigen Entwicklungsabteilungen gängige Praxis darstellen, wird eine Bewertung von zwei Punkten vergeben (Vorteile und Nachteile gleichgewichtig) [0, Z. 27 ff.]. In Jenkins sowie Azure Pipelines lassen sich die Entwicklungsumgebungen Microsoft Visual Studio Code sowie Eclipse integrieren (*Integrationsmöglichkeiten von Repositories (Kriterium K2.2)*). Die SAP CI/CD-Pipeline un-

terstützt keine der zu evaluierenden Entwicklungsumgebungen [0, Z. 94 ff.]. Somit ist es Entwicklern nicht möglich, die CI-Pipeline unmittelbar aus der Entwicklungsumgebung zu starten. Da sowohl für Azure Pipelines als auch Jenkins ausschließlich zwei der drei zu evaluierenden Entwicklungsumgebung unterstützt werden, wird eine Bewertung von drei Punkten vergeben (Vorteile überwiegen Nachteilen). SAP CI/CD wird aufgrund der fehlenden Integrationsmöglichkeiten mit null Punkten bewertet (ausschließlich Nachteile). Sowohl mit Azure Pipelines als auch Jenkins lassen sich Planungs-Tools wie Jira integrieren (*Integrationsmöglichkeiten von Planungssoftware (Kriterium K2.2)*). Für SAP CI/CD besteht diese Integrationsmöglichkeit nicht [65, Z. 101 ff.]. Demnach ist es Projektmanagern nicht möglich, den Build-Status verschiedener Backlog-Items zu begutachten. Aus diesem Grund ergibt sich eine Bewertung von vier Punkten für Azure Pipelines sowie Jenkins bzw. null Punkte für den SAP BTP-Service. In Kriterium K3 werden die **Kosten** der CI/CD-Pipelines evaluiert. Für SAP CI/CD werden Gebühren von einem Euro pro Build-Stunde fällig [0, Z. 107 ff.]. Azure Pipelines berechnet hingegen unabhängig von der Nutzungszeit 40 Euro pro Pipeline [23]. In den genannten Preisen werden Kosten für die IT-Infrastruktur, Installation, Wartung sowie Support berücksichtigt. Zwar fallen für die Jenkins-Pipeline keine Lizenzgebühren an, jedoch müssen im Gegensatz zu den SaaS-Tools weitere Kostenpositionen beachtet werden. Da Jenkins in einem On-Premise-Modell betrieben wird, müssen hierbei zunächst Investitionskosten für Hardware berücksichtigt werden. Weiterhin fallen für den On-Premise-Server laufende Betriebskosten, wie Ausgaben für Energie, Reparaturleistungen und Personal zur Wartung der IT-Infrastruktur an. Untersuchungsergebnisse des IT-Beratungshauses ExecuTech zeigen, dass On-Premise-Systeme aufgrund hoher Investitionskosten insbesondere bei einer kurzfristigen Betrachtung teurer sind [57]. Eine langfristige Perspektive führt jedoch zu einem anderen Ergebnis. Da die laufenden Betriebskosten für On-Premise-Systeme häufig geringer als die SaaS-Gebühren sind, können diese auf lange Sicht kosteneffektiver werden. CEs sind jedoch darauf ausgerichtet, flexibel und agil zu agieren, um auf sich ändernde Geschäftsanforderungen reagieren zu können. Somit müssen diese in der Lage sein, Systeme schnell auf- und abzu-

bauen. Dies würde im On-Premise-Modell kontinuierliche Investitionskosten verursachen, wobei SaaS ebenfalls auf längere Sicht die günstigere Alternative bleibt. Aus diesem Grund wird für die SaaS-Systeme Azure Pipelines und SAP CI/CD eine Bewertung von drei Punkten bzw. für Jenkins einen Punkt vergeben (Nachteile überwiegen Vorteilen). In Kriterium K4 wird die **Skalierbarkeit** der Tools evaluiert. Mit Azure Pipelines lässt sich die CI/CD-Pipeline eines Microservices horizontal skalieren. Um parallele Builds auszuführen, werden hierbei mehrere *Microsoft-hosted Agents* aktiviert. Ein Microsoft-hosted Agent ist eine von Azure verwaltete virtuelle Maschine, welche eine vorkonfigurierte Build- und Testumgebung bereitstellt [33]. Eine vertikale Skalierung kann dabei durch das Zuweisen zusätzlicher Ressourcen zu einem Microsoft-hosted Agent erreicht werden. Während eine horizontale Skalierung ebenfalls über das Hinzufügen zusätzlicher Agents ermöglicht wird, ist eine vertikale Skalierung aufgrund architektonischer Limitationen bei Jenkins nur begrenzt möglich [60]. So müssen in das bestehende System zusätzliche Hardware-Ressourcen integriert werden. Das bedeutet, dass die vertikale Skalierbarkeit auf die physischen Einschränkungen eines Servers begrenzt ist. Während horizontale Skalierung von SAP CI/CD unterstützt wird, ist eine vertikale Skalierung nicht realisierbar. Die vertikale Skalierbarkeit von CI/CD-Pipelines spielt dabei insbesondere in einem volatilen Geschäftsumfeld eine signifikante Rolle. Als eines der größten CEs stellt der Streaminganbieter Netflix über 700 Microservices bereit. Angesichts der Notwendigkeit, für jeden Microservice mindestens eine CI/CD-Pipeline bereitzustellen, würde ein nicht skalierbares Pipeline-System eine umfangreiche IT-Infrastruktur erfordern. Netflix setzt aus diesem Grund auf eine vertikal-skalierbare Container-Orchestrierungstechnologie [58]. Mit dieser werden zur Laufzeit Pipelines in Docker-Container bereitgestellt, welche je nach Bedarf auf- oder abgebaut bzw. skaliert werden können. So wird Azure Pipelines mit vier Punkten (ausschließlich Vorteile) bzw. Jenkins und SAP CI/CD mit einem Punkt bewertet (Nachteile überwiegen Vorteilen). In Kriterium K5 wird die **Performance** der Pipelines evaluiert. Zur Durchführung dieser Analyse wird ein speziell für die Arbeit entwickelter Prototyp einer CEA verwendet. Dieser besteht aus drei Microservices, welche jeweils

ein Frontend (SAP UI5), eine Service-API (SAP CAP Node) sowie eine Datenbank (SAP HANA) besitzen (s. Abb. 17). Des Weiteren ist mit den zu untersuchenden Tools für jeden Microservice eine eigene CI- sowie CD-Pipeline implementiert. Die CI-Pipeline besteht hierbei aus einem MTA-Build, statischen Lint-Code-Analysen, Backend-Unit-Tests, Frontend-Unit- sowie -Integration-Tests. Die CD-Pipeline erweitert den CI-Prozess um Sonar-Qube-Code-Analysen, End-To-End-Tests sowie einem Schritt zur Bereitstellung der Anwendung auf der SAP BTP. Im Vergleich zu den anderen zu evaluierenden Tools weist der SAP CI/CD den geringsten Funktionsumfang auf. Um eine bessere Vergleichbarkeit zu gewährleisten, ist der Aufbau der anderen Pipelines strukturell an den Funktionsumfang dieses Tools angepasst. So sind etwa für keine der Pipelines Integration-Tests, sowie Security-Scans implementiert.

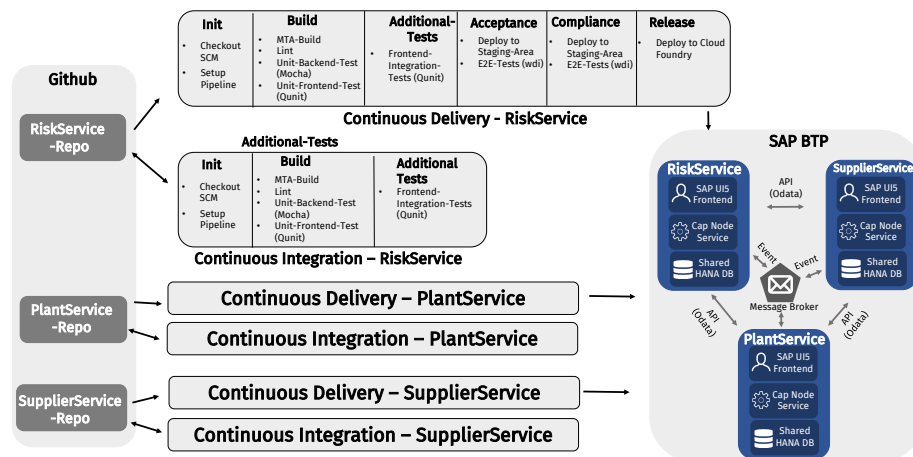


Abbildung 17: CEA-Szenario für Performance-Tests. Eigene Darstellung.

Für jedes zu untersuchende CI/CD-Tool wird dabei die Durchlaufzeit der Pipelines aller drei Microservices in Sekunden gemessen. Anschließend werden die Einzelmessungen der Services aggregiert, um für jedes zu evaluierende CI/CD-Tool eine Integration- sowie Delivery-Zeit zu erhalten (s. Anhang 24). Die Ergebnisse der Tests zeigen, dass für den vollständigen Durchlauf der CI-Prozesses aller Microservices für Azure Pipelines 600 Sekunden, Jenkins 1134 Sekunden und SAP CI/CD 1243 Sekunden benötigt werden (s. Tab. 2). Unter der in Kapitel 4.2 spezifizierten Metrik ergibt

sich im Kriterium *K5.1* eine Bewertung von vier Punkten für Azure Pipelines (bester Zeitwert). Für Jenkins sowie SAP CI/CD werden hingegen null Punkte vergeben (75 Prozent oder mehr über dem besten Zeitwert). Für die vollständige Abwicklung des CD-Prozesses werden für Azure Pipelines 2194 Sekunden, Jenkins 3409 Sekunden und SAP CI/CD 3801 Sekunden benötigt. Daraus resultiert im Kriterium *K5.2* eine Bewertung von vier Punkten für Azure Pipelines (bester Zeitwert) und einem Punkt für Jenkins bzw. SAP CI/CD (50 Prozent bis 75 Prozent über dem besten Zeitwert). Die Ergebnisse dieses Performance-Tests sind jedoch differenziert zu betrachten. Die Überprüfungen wurden ausschließlich anhand von Prototypen mit einem geringem Umfang an Programm-Code durchgeführt. Somit ist zu berücksichtigen, dass die Ergebnisse bei umfangreicheren Produktivprojekten abweichen können. Angesichts des erheblichen zeitlichen Abstands lässt sich annehmen, dass Azure Pipelines nach wie vor die beste Performance besitzt. Zudem ist zu beachten, dass Jenkins On-Premise betrieben wird und somit bei einer divergierenden Hardware auch unterschiedliche Leistungsergebnisse erzielt werden können.



		CI-Pipeline 					CD-Pipeline 						
		Init	Build	Additional Tests	Σ	Init	Build	Additional Tests	Acceptance	Compliance	Release	Σ	
Azure Pipelines	RiskService	13	135	56	204	15	139	58	279	88	152	731	
	SupplierService	12	132	58	202	14	133	52	277	87	150	713	
	PlantService	12	130	52	194	15	138	59	286	93	159	750	
	Σ				600							2194	
Jenkins	RiskService	6	253	128	387	5	245	136	468	111	179	1144	
	SupplierService	6	243	115	364	5	238	125	455	106	167	1096	
	PlantService	6	247	130	383	6	246	134	493	113	178	1169	
	Σ				1134							3409	
SAP CI/CD	RiskService	48	249	115	412	83	279	141	471	100	203	1277	
	SupplierService	48	247	119	414	79	272	135	475	89	198	1248	
	PlantService	47	251	119	417	71	282	143	468	103	209	1276	
	Σ				1243							3801	

Tabelle 2: Integration- und Delivery-Zeit in Sekunden. Eigene Darstellung.

In Kriterium K6 wird die **Flexibilität** der Services evaluiert. Sowohl mit Azure als auch Jenkins lassen sich die mit der Programmbibliothek Project Piper bereitgestellten CI/CD-Schritte (z.B. Build, Test etc.) frei nach Bedarf kombinieren und konfigurieren. SAP CI/CD besitzt dabei maßgebliche Einschränkungen. Hierbei werden Anzahl, Reihenfolge sowie Konfiguration der Schritte von dem Service vorgeschrieben. Somit ist keine flexible Implementierung der Pipelines möglich. Laut Experte

5, stellt Technologieoffenheit einen fundamentalen Wert in einer CEA dar [70, Z. 13 ff.]. So sollte die CI/CD-Pipeline verschiedene Programmier-Frameworks, Test-Tools und Cloud-Plattformen unterstützen. Damit werden bei Jenkins mehr als 1500 Plug-ins bereitgestellt, mit denen Funktionalitäten, welche über den Standard hinausgehen, integriert werden können [41]. Während mit Azure Pipelines 1400 Plug-ins bereitgestellt werden, besteht diese Möglichkeit bei SAP CI/CD nicht [24]. Weiterhin sollten die Pipelines, um schnell neue Microservices bereitzustellen, einen flexiblen modularen Aufbau ermöglichen. Für Azure Pipelines und Jenkins wird durch die Verwendung von Project Piper das Shared-Library-Konzept realisiert (s. Abb. 18). Damit ist es möglich vorimplementierte Funktionalitäten, wie z.B. standardisierte Build, Deploy oder Test-Schritte in einer gemeinsamen Bibliothek zu konsolidieren. Eine weitere Möglichkeit, um die Komplexität der Bereitstellung zu reduzieren, besteht in der Wiederverwendung ganzer Pipelines. Dieses Konzept wird verwendet, wenn sämtliche Microservices einen homogenen Bereitstellungszyklus durchlaufen. Durch die Wiederverwendung ganzer Pipelines kann somit sichergestellt werden, dass diese auf analoge Weise kompiliert, verpackt und bereitgestellt werden. Bei Jenkins und Azure lässt sich dies durch die Einbindung mehrerer Repositories in eine Pipeline realisieren. Bei SAP CI/CD wird ebenfalls ein Shared-Library-Konzept unterstützt. So werden die templatebasierten Schritte als standardisierte Elemente einer CI/CD-Pipeline ausgeliefert. Die Wiederverwendung ganzer Pipelines für unterschiedliche Microservices ist hingegen nicht möglich.

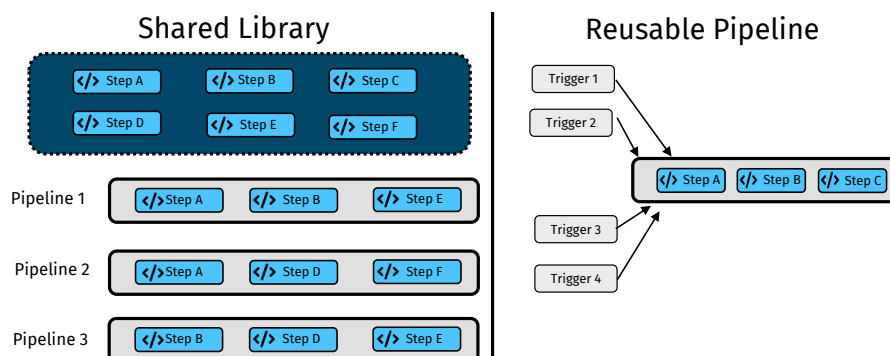


Abbildung 18: Modularer Aufbau einer CI/CD-Pipeline. In Anlehnung an Codefresh [31].

Während der SAP CI/CD-Service das Shared-Library-Konzept unterstützt, ist eine flexible Implementierung der Pipelines, die Einbindung von Plug-ins, sowie die Wiederverwendung von CI/CD-Pipelines nicht möglich. Deshalb wird für den SAP CI/CD-Service eine Bewertung von einem Punkt vergeben (Nachteile überwiegen Vorteilen). Da Jenkins und Azure Pipelines bei den zu untersuchenden Entscheidungskriterien nur Vorteile bieten, werden jeweils vier Punkte vergeben. In Kriterium K8 wird der **Support** der CI/CD-Tools evaluiert. Azure Pipelines stellt einen umfangreichen *administrativen Support* bereit (Kriterium K7.1). So besteht für Kunden die Möglichkeit einen Premium-Support in Anspruch zu nehmen [61]. Dabei handelt es sich um eine kostenpflichtige Leistung, welche den Kunden unmittelbaren Zugang zu technischem Support und Beratung gewährt. Die Unterstützung beinhaltet Installation bzw. Konfiguration von Pipelines, Integration einer vorhandenen CI/CD-Toolchain (Tests, Repositories etc.) einschließlich der Optimierung der Pipeline-Performance. Zudem wird über das Azure-Support-Center ein kostenfreies Support-Ticket-System für Nicht-Premium-Nutzer angeboten. Diese Kanäle können verwendet werden, um allgemeine technische Probleme des CI/CD-Services zu melden. Weiterhin wird von Azure eine umfangreiche Dokumentation sowie Schulungsmaterialien, einschließlich Online-Tutorials, Videos und Webinare bereitgestellt. Der SAP CI/CD-Service stellt über den Drittanbieter Service Now ebenfalls ein kostenfreies Ticket-System zur Verfügung. Für spezifische Fragestellungen und individuelle Unterstützung besteht für Kunden die Möglichkeit, sich an die technische Beratung der SAP wenden [0, Z. 113 ff.]. Wie auch bei Azure Pipelines stehen für SAP CI/CD Schulungs- und Dokumentationsunterlagen zur Verfügung. Für Jenkins sind diese Informationsmaterialien ebenfalls vorhanden, jedoch ist der administrative Support aufgrund des Open-Source-Charakters im Vergleich zu den anderen Lösungen begrenzt. Dies ist insbesondere auf das Fehlen eines unmittelbaren Supports durch einen Service-Anbieter zurückzuführen. Ebenso spiegelt sich dieser Aspekt in der Verfügbarkeit von Updates wider. Viele der Funktionalitäten von Jenkins werden über Open-Source-Plug-ins bereitgestellt. Experte 6 merkt an, dass hierbei stets das Risiko besteht, dass die Wartung einzelner Plug-ins in Zukunft vernachlässigt werden könnte [66, 29 ff.]. Aus den aufgeführten Aspekten ergibt sich eine Bewertung

von vier Punkten für Azure Pipelines sowie dem SAP CI/CD-Service (ausschließlich Vorteile). Für Jenkins wird eine Bewertung von einem Punkt vergeben (Nachteile überwiegen). Aufgrund des Open-Source-Charakters ist für die Jenkins-Pipeline ein umfassender *Community-Support* vorhanden (Kriterium K7.2). Jenkins ist dabei auf zahlreichen hochfrequentierten Foren vertreten. So wurden auf Stack-Overflow bereits 112000 Posts zur Jenkins-Pipeline veröffentlicht [48]. Dadurch wird den Anwendern Zugang zu einer breiten Palette an Ressourcen ermöglicht. Experte 4 merkt an, dass insbesondere die Gamifizierung dieser Plattformen zu einer schnellen und effektiven Unterstützung durch Spezialisten beiträgt. Auch Azure Pipelines sowie der SAP CI/CD sind ebenfalls auf öffentlichen Community-Foren vertreten, jedoch ist der dort publizierte Inhalt im Vergleich zu Jenkins signifikant geringer. Das liegt an der Tatsache, dass die genannten Pipelines als SaaS-Lösungen konzipiert sind und somit weniger Raum für fragebedürftige Anpassungen bestehen. So sind für Azure Pipelines 25000 Post und für SAP CI/CD lediglich 26 Einträge auf Stack-Overflow veröffentlicht [49][47]. Gemäß der in Kapitel 4.2 festgelegten Metrik wird eine Bewertung von vier Punkten für Jenkins (höchste Blog-Post-Anzahl) bzw. von einem Punkt für Azure Pipelines und SAP CI/CD vergeben (0 Prozent bis 25 Prozent unter dem höchsten Wert). In Kriterium K8 wird die **Sicherheit** der CI/CD-Tools evaluiert. Jenkins bietet flexible Möglichkeiten zur Authentifizierung. Neben einer integrierten Benutzerverwaltung lassen sich über Jenkins ebenfalls Drittanbieterdienste wie Github, Google, BitBucket oder SAP-Single-Sign-On (SAP-SSO) einbinden [40]. Des Weiteren ist bei Jenkins die Implementierung eines Rollenkonzeptes möglich. Dabei können Rollen und Berechtigungen erstellt und bestimmten Benutzern zugewiesen werden. Damit wird sichergestellt, dass kritische Konfigurationen ausschließlich von Spezialisten vorgenommen werden [51]. Azure Pipelines und SAP CI/CD unterstützen ebenfalls Authentifizierungsmöglichkeiten. Die Implementierung eines Rollenkonzeptes kann im SAP CI/CD-Service jedoch nicht vorgenommen werden. Aufgrund der Integration eines Plug-in-Ökosystems erhöht sich im Kontext der Systemsicherheit für Jenkins das Risiko unbefugter Zugriffe. Da in der Jenkins-Community jeder Entwickler Plug-ins veröffentlichen kann, besteht die Möglichkeit,

dass Sicherheitsrisiken mit diesen Erweiterungen einhergehen. Bei Azure Pipelines ist die Einbindung von Plug-ins ebenfalls möglich. Da hierbei jedoch ausschließlich validierte auf dem Azure Marktplatz bereitgestellte Plug-ins integrierbar sind, fällt das korrespondierende Sicherheitsrisiko deutlich geringer aus. IT-Sicherheit gehört zum Kerngeschäft von Cloud-Anbietern wie Microsoft oder SAP. Aus diesem Grund verfügen diese über erfahrenes Sicherheitspersonal. Somit lässt sich schlussfolgern, dass sowohl Azure Pipelines als auch SAP CI/CD ein hohes Niveau an Systemsicherheit bieten. Da Jenkins im On-Premise-Modell bereitgestellt wird, benötigt ein Unternehmen eigenes Sicherheitspersonal. Insbesondere in kleineren Organisationen kann es aufgrund finanzieller Einschränkungen jedoch vorkommen, dass diese kein hochqualifiziertes Sicherheitspersonal bereitstellen können. Jedoch bietet sich im On-Premise der Vorteil, dass Unternehmen vollständige Kontrolle über die Systemkonfiguration behalten. So ist diesem möglich, gezielte Maßnahmen zur Verbesserung der Sicherheit zu ergreifen. Im Gegensatz dazu hängt die Sicherheit von SAP CI/CD bzw. Azure Pipelines in erheblichem Maße von den Cloud-Anbietern ab. Aufgrund der situativen Gegebenheiten werden Vor- und Nachteile der On-Premise- bzw. Cloud-Bereitstellung als gleichwertig betrachtet. Daraus resultiert sowohl für Azure Pipelines als auch für den SAP CI/CD-Service eine Bewertung von zwei Punkten. Aufgrund der zusätzlichen Sicherheitsrisiken im Kontext des Plug-in-Ökosystems wird für Jenkins eine Bewertung von einem Punkt vergeben (Nachteile überwiegen Vorteilen). In Kriterium K9 wird die **Benutzerfreundlichkeit** der Pipelines evaluiert. Da Azure Pipelines und SAP CI/CD als SaaS-Lösung vertrieben werden, bieten diese hinsichtlich der *Installation und Wartung (Kriterium K9.2)* einige Vorteile. Bei cloudbasierten Anwendungen entfällt die Notwendigkeit CI/CD-Systeme zu installieren und einzurichten. Auch die Wartung der IT-Infrastruktur liegt in Verantwortung der Cloud-Anbieter. Dadurch werden IT-Spezialisten zeitlich entlastet, wodurch Fachpersonal verstärkt auf die Kernkompetenzen des Unternehmens fokussiert werden können. Da Jenkins in einer On-Premise-Umgebung betrieben wird, obliegt die Verantwortung der Installation und Wartung den Unternehmen selbst. Daraus resultiert eine Bewertung von vier Punkten für Azure Pipelines und SAP

CI/CD (ausschließlich Vorteile) bzw. von null Punkten für Jenkins (ausschließlich Nachteile). In Kriterium K9.2 wird die *intuitive Bedienbarkeit* der Tools evaluiert. Azure Pipelines und SAP CI/CD bieten eine webbasierte Benutzeroberfläche, über welche Build-Prozesse, Systemkonfigurationen und Pipelines eingesehen und angepasst werden können. Die Implementierung der Pipeline erfolgt dabei jedoch über Code. Während für Azure Pipelines eine YAML-basierte Syntax verwendet wird, werden CI/CD-Pipelines in Jenkins mit Groovy implementiert. Unter Verwendung von Project Piper kann eine Pipeline jedoch mit minimalem Aufwand an Code erstellt werden. Im Gegensatz dazu benötigt der SAP CI/CD-Service keine manuelle Implementierung über Code. So können Anwender mithilfe einer webbasierten Benutzeroberfläche die gesamte Implementierung einer Pipeline konfigurieren. Dies ist dabei insbesondere für CEs von wesentlicher Bedeutung. Bei der Bereitstellung neuer Microservices ist ggf. die Implementierung einer neuen Pipeline erforderlich. Eine intuitive Bedienbarkeit bietet dabei erhebliche Vorteile, um eine schnelle Erstellung von Pipelines zu gewährleisten. Aus diesem Grund erhält SAP CI/CD eine Bewertung von vier Punkten (ausschließlich Vorteile). Obwohl Project Piper grundsätzlich zur Vereinfachung der Implementierung beitragen kann, benötigt es einer aufwendigen Programmierung von Anforderungen, welche über den Bibliotheksstandard hinausgehen. Aus diesem Grund wird für Azure Pipelines sowie dem SAP CI/CD-Service eine Bewertung von einem Punkt vergeben (Nachteile überwiegen Vorteilen).










	B: Bewertung gB: gewichtete Bewertung	Globale Gewichtung pro hundert	Azure Pipelines		Jenkins		SAP CI/CD	
			B	gB	B	gB	B	gB
 K1 Funktionalität		0,1728	-	-	-	-	-	-
K1.1 Tests		0,0548	4	0,2192	4	0,2192	3	0,1644
K1.2 Code-Analysen		0,0212	4	0,0848	4	0,0848	1	0,0212
K1.3 Build		0,0380	4	0,1520	4	0,1520	2	0,0760
K1.4 Deploy		0,0390	3	0,1770	3	0,1770	3	0,1770
K1.5 Monitoring		0,0198	4	0,0792	3	0,0594	1	0,0198
 K2 Integrationsmöglichkeiten		0,1580	-	-	-	-	-	-
K2.1 Integration in Repository		0,0796	4	0,3076	3	0,2307	2	0,1538
K2.2 Integration in Entwicklungsumgebung		0,0444	3	0,1332	3	0,1332	0	0,0000
K2.3 Integration in Planungs-Software		0,0340	4	0,1360	4	0,1360	0	0,0000
 K3 Kosten		0,0963	3	0,2889	1	0,0963	3	0,2889
 K4 Skalierbarkeit		0,1160	4	0,4640	1	0,1160	1	0,1160
 K5 Performance		0,0790	-	-	-	-	-	-
K5.1 Integration-Zeit		0,0574	4	0,2296	0	0,000	0	0,000
K5.2 Delivery-Zeit		0,0216	4	0,0864	1	0,0216	1	0,0216
 K6 Flexibilität		0,1185	4	0,4740	4	0,4740	1	0,1185
 K7 Support		0,0815	-	-	-	-	-	-
K7.1 Administrativer Support		0,0290	4	0,1160	1	0,0290	4	0,1160
K7.2 Community Support		0,0525	1	0,0525	4	0,2100	1	0,0525
 K8 Sicherheit		0,1259	2	0,2518	1	0,1259	2	0,2518
 K9 Benutzerfreundlichkeit		0,0519	-	-	-	-	-	-
K9.1 Installation und Wartung		0,0222	4	0,0888	0	0,0000	4	0,0888
K9.2 Intuitive Bedienbarkeit		0,0296	1	0,0296	1	0,0296	4	0,1184
			4,0642		2,2947		1,7874	

Tabelle 3: Ergebnistabelle zum AHP. Eigene Darstellung.

Unter Berücksichtigung der zu untersuchenden Fragestellung ergibt sich eine Gesamtbewertung von 4,0642 für Azure Pipelines, 2,2947 für Jenkins und 1,7874 für SAP CI/CD (s. Tab. 3). Somit kann Azure Pipelines auf Grundlage des in der Arbeit entworfenen Entscheidungs-Framework, als das optimale CI/CD-Tool betrachtet werden.

5 Entwicklung einer Handlungsempfehlung

In einem wettbewerbsintensiven Umfeld reicht es häufig nicht aus die Bedürfnisse der Anwender zu verstehen und innovative IT-Services zu entwickeln. Stattdessen sollten Unternehmen eine höhere Priorität darauf legen, Dienste im Vergleich zur Konkurrenz schneller bereitzustellen. Infolgedessen verwenden 62 Prozent aller Unternehmensentwickler CI/CD-Tools innerhalb ihrer Bereitstellungsprozesse. Die kontinuierliche Integration und Bereitstellung von Software bietet dabei insbesondere für CEs erhebliche Vorteile. Obwohl einzelne Module einer CEA grundsätzlich isoliert voneinander betrieben werden, können Abhängigkeiten zwischen diesen entstehen. Im Kontext eines Composable-ERP-Systems könnte es etwa erforderlich sein, dass die in einem Vertriebsmodul abgewickelten Verkäufe in das Finanzwesen übertragen werden müssen. Insbesondere bei der Zusammenarbeit vieler Entwickler kann diese Integration zu erheblichem Koordinationsaufwand führen. Wenn Teams nicht in der Lage sind effektiv miteinander zu kommunizieren, besteht das Risiko, dass Konflikte in der gemeinsamen Code-Basis entstehen. Um dieser Herausforderung zu begegnen, bietet sich die Implementierung einer CI/CD-Pipeline. Dabei wird der lokale Quell-Code der Entwickler kontinuierlich und automatisiert mit dem Hauptzweig des Repositories zusammengeführt und mit dem bestehenden Code getestet. Statt Code-Reviews und Validierungen erst in einer späten Phase des Softwareerstellungszykluses abzuwickeln (*Shift-Right*), werden Tests bereits während der Entwicklung durchgeführt (*Shift-Left*). Nach Erstellung einer Funktionalität bekommen Entwickler über verschiedene Kommunikationskanäle (z.B. E-Mail) unmittelbares Feedback und können den Service somit optimieren, bis dieser den Produktstandards des Unternehmens entspricht. Dieser Ansatz zielt darauf ab, eine frühzeitige Identifikation und Behebung von Fehlern zu erleichtern. Darüber hinaus ermöglichen CI/CD-Pipelines den Aufbau standardisierter Testinfrastrukturen. Damit können neue Features automatisiert in Systemumgebungen getestet werden, bei welchen Betriebssysteme, Systembibliotheken, Abhängigkeiten und Konfigurationseinstellungen an die Produktion angepasst sind. Zum einen können somit potenzielle

Fehler erkannt werden, welche ansonsten im produktiven Betrieb auftreten würden. Darüber hinaus entfällt durch das automatische Abwickeln dieser Prozesse, ebenfalls ein manuelles Aufsetzen der Testinfrastruktur. So können CEs ihre Ressourcen vermehrt auf das Kerngeschäft, also die Entwicklung neuer ERP-Funktionalitäten, konzentrieren. Da für jeden Microservice i.d.R. eine CI- sowie CD-Pipeline betrieben wird, gestaltet es sich insbesondere bei einer umfassenden CEA als herausfordernd, den gesamten Bereitstellungsprozess zu überwachen. Dabei Abhilfe schaffen sollen Monitoring-Dashboards, mit welchen die gesamten Pipeline-Workflows eines ERP-Systems in einem zentralen Tool überwacht werden können. Die dabei analysierten Metriken werden ebenfalls in einen historischen Datenkontext eingeordnet, womit Entwickler-Teams Trends in der Code-Qualität identifizieren können. Dies ermöglicht eine Ermittlung von Mustern in vorliegenden Fehlerdaten, womit Problembereiche der bestehenden Code-Basis erkannt und deren Korrektur priorisiert werden kann. Der von Panorama Consulting Solutions veröffentlichte ERP Report 2019 zeigt, dass ERP-Systeme aufgrund der Geschwindigkeit mit welcher sich Unternehmenstechnologien und Marktdynamiken entwickeln, zukunftsorientiert sein müssen [12]. Statt auf wiederholte kostspielige Migrationen zurückgreifen zu müssen, sollten Unternehmen daher bestehende Systeme durch gezielte Erweiterungen und Modifikationen optimieren. So ist es von essenzieller Bedeutung, dass nicht nur das Testen, sondern ebenfalls die Bereitstellungen in die Produktionssysteme kontinuierlich und automatisiert durchgeführt werden. Durch das unmittelbare Kompilieren, Validieren und Versionieren bei der Integration neuen Codes, kann zu jedem Zeitpunkt eine Anwendungsversion bereitgestellt werden, welche für eine Veröffentlichung geeignet ist. Unter Zuhilfenahme eines effektiven CI/CD-Prozesses kann somit ein mehrfaches tägliches Ausrollen der Dienste ermöglicht werden. Dies führt zur Verkürzung des *Time-To-Values*, also dem Intervall bis der entwickelte IT-Service den ersten Kundennutzen herbeiführt. Obwohl die Zerlegung großer Epics in kleine kontinuierliche Feature-Releases dazu führen kann, dass der Kundennutzen im Vergleich zur vollständigen Einführung abgeschwächt wird, bietet eine solche Früheinführung den Vorteil, dass ERP-Funktionalitäten eingeführt werden, welche

einen Vorsprung gegenüber Konkurrenten ermöglichen [6, S. 9]. Trotz erfolgreicher Abwicklung aller Validierungen, besteht die Möglichkeit, dass fehlerhafter Code von der CI/CD-Pipeline auf das Produktionssystem geladen wird. Die Verwendung kleiner Code-Bereitstellungszyklen erleichtert Entwickler jedoch die Identifikation dieser Probleme und ermöglicht somit eine zügige Fehlerbehebung. Eine korrespondierende Erkenntnis geht ebenfalls aus dem State of CD Report 2022 hervor. So soll eine schnelle Bereitstellung von IT-Services, ebenfalls in einer zügigeren Behebung von Fehlern resultieren (s. Tab. 4).

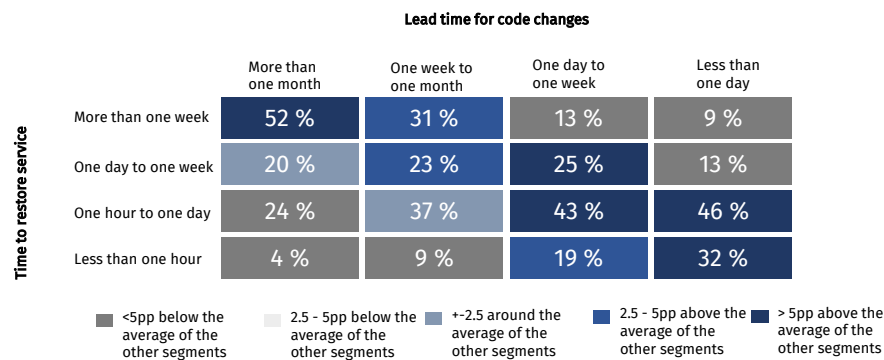


Tabelle 4: Korrelation von Vorlaufzeit und Fehlerrate bei der Bereitstellung.
In Anlehnung an Berry.

Entschließt sich ein CE dazu Bereitstellungsprozesse zu automatisieren, sollte dieses strategisch dabei vorgehen. So empfiehlt Experte 5, Softwarearchitekt des SAP DTS, ein schrittweises Implementieren der Pipeline. Dabei sollte zunächst Erfahrung mit der Automatisierung einfacher isolierter Prozesse gesammelt werden [70, Z. 8 ff.]. Anschließend können verbleibende, zur Bereitstellung von Software benötigten Schritte, in einem kontrollierten Tempo in die Pipeline eingebunden werden. Bei der Einführung von CI/CD ist ebenso entscheidend, dass Unternehmen spezifische Anforderungen abdeckende Tools verwenden. Dafür wurde im Rahmen dieser Arbeit ein Entscheidungs-Framework entworfen. Mit diesem wurde evaluiert, welches Pipeline-Tool zur Automatisierung der CI/CD-Prozesse für CEA den größten Mehrwert birgt. Unter Berücksichtigung der in Kapitel 4.4 abgewickelten Analyse, kann Azure Pipelines als das optimale CI/CD-Tool angesehen werden. Aus diesem Kontext lassen sich für dieses Pipeline-Tool einige Vorteile ableiten. Azure Pipelines

unterstützt die Programmbibliothek Project Piper, mit welcher essenzielle Schritte für das Bauen, Testen und Bereitstellen von SAP-CAP-Node- und SAP-UI5-Anwendungen ausgeliefert werden. Die Bereitstellung vorimplementierter Schritte stellt sicher, dass sämtliche Compliance-Standards der SAP, wie Sicherheits- und Datenschutzbestimmungen, Branchenvorschriften oder interne Richtlinien, während des Bereitstellungsprozesses eingehalten werden. Obwohl die vorliegende Arbeit zur Beratung externer Kunden konzipiert ist, welche nicht verpflichtet sind, diese Richtlinien einzuhalten, sind diese oft in kritischen Sektoren, wie der Finanz- oder Versicherungsindustrie, tätig, bei welchen ebenfalls strikte Regularien gelten. Durch die Bereitstellung dieser standardisierten Bibliothek entfällt für Unternehmensentwickler somit die Notwendigkeit einer zeitaufwendigen Implementierung dieser Schritte. Neben der Möglichkeit, dass CEs auf diese Weise in der Lage sind, neue Services schnell bereitzustellen, erleichtert eine zentrale Bibliothek ebenfalls Aktualisierung und Pflege einzelner Pipeline-Schritte. Des Weiteren werden durch die Programmbibliothek Project Piper ebenfalls umfassende Bereitstellungsfunktionalitäten zur Verfügung gestellt. So umfasst diese neben vorimplementierten Schritten zum Ausrollen von Anwendungen auf der SAP BTP, ebenfalls ein Blue/Green-Deployment. Das Blue-Green-Deployment stellt eine effektive Möglichkeit dar, Software unter Produktionsbedingungen zu testen und kritische Ausfallszeiten bei der Bereitstellung von Software zu vermeiden. Dies spielt insbesondere dann eine wichtige Rolle, wenn Fehler in einem Microservice eines ERP-Systems auftreten, jedoch das herkömmliche Bereitstellen einer neuen Anwendungsversion aufgrund der durch die Initialisierung bedingten Ausfallzeit, zu unflexibel ist. Ein weiteres in Azure Pipelines kompatibles Bereitstellungskonzept ist das Multi-Cloud-Deployment. Damit können einzelne Module, wie CRM- oder SCM-Komponenten, auf unterschiedlichen Cloud-Plattformen ausgerollt werden. Neben einer Bereitstellung auf etablierten Cloud-Plattformen wie Google Cloud oder Amazon Web Services ermöglicht Azure Pipelines in diesem Kontext ebenfalls eine nahtlose Integration mit den eigenen Cloud-Diensten. Dieser Integrationsvorteil erstreckt sich ebenfalls auf andere Services, welche innerhalb des Azure-Ökosystems bereitgestellt werden. So umfasst dieses

darüber hinaus Dienste, wie eine Entwicklungsumgebung, ein Projektmanagement- und Monitoring-Tool sowie ein Artefakt-Repository, welche mit geringem Aufwand in die CI/CD-Pipeline integriert werden können. Laut Experte 5 stellt Technologieoffenheit einen wichtigen Aspekt für CEs dar [70, Z. 8 ff.]. Obwohl die Evaluation dieser Arbeit auf SAP-Technologien beschränkt ist, besteht die Möglichkeit, dass sich ein CEs dazu entscheidet einzelne Services auf anderen Programmiersprachen aufzubauen. Azure Pipelines stellt im Standard bereits unzählige Build-Tools sowie Test-Frameworks zur Verfügung. Sollte dieser jedoch nicht ausreichend sein, können mit Azure Pipelines ebenfalls Docker-Container aufgebaut werden. Mit diesen besteht die Möglichkeit Abhängigkeiten, wie Softwaretreiber, in einer isolierten Infrastruktur zu installieren, ohne von der Laufzeitumgebung der Pipeline abhängig zu sein. Azure Pipelines kann darüber hinaus zur Vereinfachung des Entwicklungsprozesses beitragen. So ist es Entwicklern möglich, Tests der Integration-Pipeline unmittelbar aus der Visual-Studio-Code-Umgebung auszuführen, ohne, dass Features zunächst in das Repository geladen werden müssen. Des Weiteren können mit dem CI/CD-Tool Pipeline-Implementationen sowohl lokal aus den Entwicklungsumgebungen, als auch zentral in dem Azure-Dashboard bearbeitet werden. Diese Praktikabilität ergibt sich ebenfalls bei der Ausführung der CI/CD-Pipelines. Im Falle von Fehlschlägen einzelner Schritte, besteht die Möglichkeit, dass nur fehlerhafte Schritte erneut ausgeführt werden. Auf diese Weise lassen sich bei temporären Problemen oder externen Abhängigkeiten, wie Ausfällen von Drittanbietern, Zeit und Rechenressourcen einsparen. Laut Experte 3 spielt dies insbesondere bei der Bereitstellung umfangreicher ERP-Services eine wichtige Rolle, da Pipeline-Schritte wie Code-Analysen mitunter über fünf Stunden in Anspruch nehmen können [69, Z. 20 ff.]. Dies erwies sich ebenfalls im One-Strike Programm als einen bedeutenden Aspekt. In diesem Zusammenhang hat die SAP Maßnahmen ergriffen, um die Anzahl der Cloud-Provider, von welchen Dienste bezogen werden, zu reduzieren. Im Rahmen dieser Konsolidierung wurden Frontend-End-Pipelines für das S/4HANA-Core, welche zuvor auf Jenkins gehostet wurden, zu Azure migriert. Ein weiterer wesentlicher Aspekt für diesen Wechsel, waren Kostenüberlegungen. Durch die Nut-

zung einer SaaS-basierten CI/CD-Lösung können erhebliche Einsparungen bei Aufwandspositionen, Hardwareinvestitionen oder Wartungs- und Support-Kosten erzielt werden. Dieser Aspekt birgt insbesondere einen erheblichen Mehrwert für die CEA. Da Unternehmen mit dieser Architektur bestrebt sind, schnell auf disruptive Marktveränderungen zu reagieren, müssen diese in der Lage sein, Services wie CI/CD-Pipelines schnell auf- und abbauen bzw. skalieren zu können. Während in einem On-Premise-Modell hierfür ggf. hohe Investitionen erforderlich sind, werden Rechenressourcen bei Azure unmittelbar bereitgestellt und in Abhängigkeit der Nutzung bepreist (*Pay-as-you-go*). Da die Bereitstellung von Services zum Kerngeschäft von Microsoft gehört, wird kontinuierlich in die Aktualisierung und Verbesserung der Azure-Infrastruktur investiert. Neben der Verwendung neuester Hardware wird die hohe Leistungsfähigkeit von Azure Pipelines ebenfalls durch das Bereitstellen von Optimierungsmechanismen sichergestellt. Dazu gehört etwa das parallele Ausführen verschiedener Pipeline-Schritte, was insbesondere bei umfangreichen Testvorgängen einen hohen Mehrwert birgt. Ferner besteht die Möglichkeit der Verwendung von Caching-Mechanismen. Damit können Ressourcen, wie Artefakte oder Daten, welche während des Pipeline-Prozesses heruntergeladen werden, für weitere CI/CD-Workflows auf der Cloud zwischengespeichert werden. Laut Experte 4, haben diese Konzepte dazu beitragen, dass der CI/CD-Prozess der internen Standardentwicklung um 35 Prozent beschleunigt wurde [65, Z. 58 ff.]. Obwohl Azure Pipelines die für eine CEA essenziellen Funktionalitäten bereitstellt, besteht die Möglichkeit, dass Unternehmen aufgrund situativer Gegebenheiten, Jenkins oder SAP CI/CD verwenden. Dafür ausschlaggebende Gründe werden in dem in Abb. 19 dargestellten Entscheidungsbaum aufgeführt. Laut Experte 1 sollte SAP CI/CD insbesondere von Kunden verwendet werden, welche über wenige Microservices verfügen und somit keine komplexen Anforderungen an den Bereitstellungsprozess besitzen [0, Z. 58 ff.]. Auch wenn mit SAP CI/CD für SAP CAP Node sowie SAP UI5 essenzielle Pipeline-Schritte bereitgestellt werden, sind Unternehmen mit diesem Tool in ihren Funktionalitäten eingeschränkt. Dies ist maßgeblich darauf zurückzuführen, dass das CI/CD-Tool ausschließlich eine Konfiguration und keine Implementierung

der Pipelines zulässt. Laut Experte 1 ist dieses Tool jedoch insbesondere für Kunden vorgesehen, welche aus der traditionellen On-Premise-Umgebung auf die SAP BTP umsteigen [0, Z. 58 ff.]. Hierbei besitzen Entwicklungs-Teams i.d.R. nicht über erforderliche DevOps-Kenntnisse, da diese zunächst mit dem Lernen allgemeiner Cloud-Konzepte, wie z.B. Programmier-Frameworks, beschäftigt sind. So soll das im Jahr 2020 veröffentlichte Tool kontinuierlich erweitert werden, um den wachsenden Anforderungen der Nutzer gerecht zu werden.

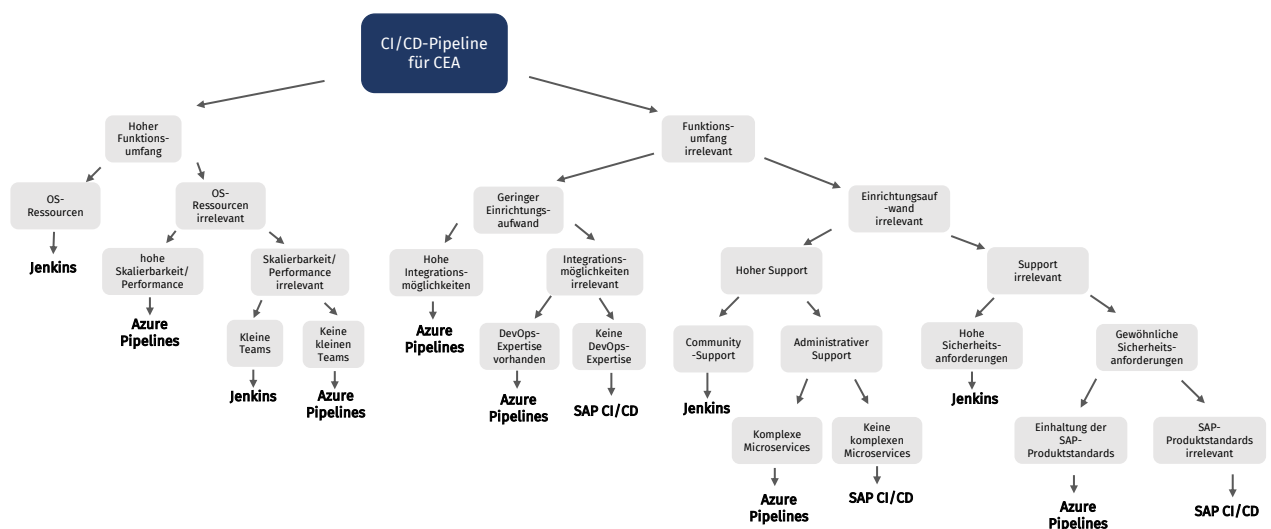


Abbildung 19: Entscheidungsbaum für die Wahl eines CI/CD-Pipeline-Tools.
Eigene Darstellung.

Weiterhin empfiehlt sich dieses Tool für CEs, welche bereits ein umfangreiches Produktportfolio der SAP beziehen und auch zukünftig auf SAP-Technologien setzen möchten. So könnten SAP-Kunden, welche neben SAP CI/CD ebenfalls andere Services des Unternehmens beziehen, potenziell von niedrigeren Gebühren für das Tool profitieren. Eine weitaus höhere Flexibilität ergibt sich durch die Verwendung von Jenkins. Da dieses CI/CD-Tool On-Premise bereitgestellt wird, besitzt ein Unternehmen volle Kontrolle über die Gestaltung des Pipeline-Tools und kann dieses somit auf die Bedürfnisse der eigenen System-Architektur ausrichten. Dies ermöglicht eine flexible Anpassung der Infrastruktur-Komponenten, wie Server- und Netzwerkmodule, Sicherheitsprotokolle oder dem Datenmanagement. Somit bietet sich eine On-Premise-Pipeline insbesondere in Branchen an, in welchen Datensicherheit

und Compliance-Anforderungen hohe Priorität besitzen. Dazu gehören etwa Finanzdienstleistungsinstitute, das Gesundheitswesen oder die öffentliche Verwaltung. Ein weiterer Grund für die Verwendung von Jenkins besteht darin, von den im Standard bereitgestellten Funktionalitäten unabhängig zu sein. Durch den Open-Source-Charakter des Tools, stehen den Entwicklern zahlreiche externe Ressourcen zur Verfügung. Dies können etwa von der Community entwickelte Plug-ins darstellen, welche in das Pipeline-System integriert werden können. Darüber hinaus haben DevOps-Spezialisten Zugang auf eine Vielzahl in Internet-Foren veröffentlichte Informationen. Dies unterstützt Entwickler dabei, Expertise, welche zur Implementierung einer maßgeschneiderten Pipeline benötigt wird, zu erlangen. Experte 4 bemerkt, dass Jenkins ausschließlich von kleinen Entwicklungsteams, welche mit einer geringen Anzahl an Technologien arbeiten, verwendet werden sollte [65, Z. 58 ff.]. Dies lässt sich darauf zurückführen, dass durch eine Aktualisierung diverse Plug-ins mit der neuen Jenkins-Version inkompatibel sein könnten. Folglich tendieren große Entwicklungsprojekte, welche eine Vielzahl heterogener Plug-ins verwenden dazu, eine Aktualisierung möglichst lange hinauszuzögern. Somit sollte darauf geachtet werden, dass dieses Pipeline-System ausschließlich in kleinen Entwicklungsprojekten mit wenig Abhängigkeiten verwendet wird.

6 Schlussbetrachtung

6.1 Fazit und kritische Reflexion

Die CEA ist ein von dem Analystenhaus Gartner veröffentlichtes IT-Konzept, welches darauf abzielt, Agilität, Skalierbarkeit und Anpassungsfähigkeit in Organisationen zu fördern. Diese Architektur basiert auf unabhängigen und wiederverwendbaren Komponenten, welche über APIs zu einem Gesamtsystem konsolidiert werden. Im Kontext eines Composable-ERP-Systems können somit Module wie das Finanzwesen, der Vertrieb oder die Beschaffung in einzelne Services ausgelagert und nach Bedarf hinzugefügt oder entfernt werden. Um individuellen Unternehmensanforderungen gerecht zu werden, besteht die Möglichkeit, modulare Komponenten sowohl durch Eigenentwicklung als auch durch Erweiterung zu adaptieren. Um Effizienz und Anpassungsfähigkeit hierbei vollständig auszuschöpfen, ist es unerlässlich, dass diese Bausteine schnell bereitgestellt und in das bestehende System integriert werden. Dies lässt sich durch den Einsatz von CI/CD-Pipelines realisieren. Diese automatisieren den Prozess der kontinuierlichen Integration und Bereitstellung von IT-Services. Damit wird Code in regelmäßigen Abständen in ein gemeinsames Repository überführt, automatisch auf Fehler überprüft und in die Produktionsumgebung bereitgestellt. Das SAP DTS empfiehlt dabei eine Auswahl zwischen drei verschiedenen Tools. Dazu gehören Azure Pipelines, Jenkins und SAP CI/CD. Für CEAs ergeben sich im Vergleich zur traditionellen IT-Architektur jedoch divergierende Anforderungen an den Bereitstellungsprozess. Deshalb wurde im Rahmen dieser Arbeit ermittelt, welches CI/CD-Tool für CEA den größten Mehrwert birgt. Zu diesem Zweck wurde ein Entscheidungs-Framework auf Grundlage des AHP-Verfahrens entworfen, anhand welchem die CI/CD-Tools evaluiert wurden. Nach Auswertung der Analyse wurde festgestellt, dass Azure Pipelines in diesem Kontext das optimale Modell darstellt. Dies ist insbesondere auf die hohe Flexibilität und Skalierbarkeit des Tools zurückzuführen. Da Azure Pipelines eine Cloud-Lösung ist, können die Rechenressourcen des Pipeline-Systems dynamisch an die spezifischen Anforderungen der Teams angepasst werden. Um eine schnelle und effiziente

Bereitstellung der IT-Services zu ermöglichen, können Kapazitäten während Stoßzeiten somit kosteneffektiv angepasst werden. In Bezug auf Performance erweist sich Azure Pipelines im Vergleich zu anderen CI/CD-Tools als besonders leistungstark. Dies wird sowohl durch die Verwendung neuester Hardware-Technologien als auch durch den Einsatz von Mechanismen wie dem parallelen Ausführen von Pipeline-Schritten oder Caching erreicht. Mit Azure Pipelines wird ebenfalls die von der SAP veröffentlichte Programmbibliothek Project Piper unterstützt. Damit werden für SAP-Technologien benötigte vorimplementierte Pipeline-Schritte ausgeliefert. Dies ermöglicht, dass CEs Bereitstellungs-Workflows ressourcenoptimiert und standardisiert implementieren können. Des Weiteren werden mit Azure Pipelines eine Vielzahl von Plattformen, Programmiersprachen und Test-Frameworks unterstützt. Dies hilft die von CEs angestrebte Technologieoffenheit aufrechtzuerhalten. Trotz der allgemeinen Vorteile von Azure Pipelines, sind bei der Auswahl geeigneter Bereitstellungs-Tools ebenfalls „K.O.-Kriterien“ zu berücksichtigen. Da Pipelines mit Azure implementiert werden müssen, erfordert dieses Tool einen hohen Grad an DevOps-Expertise. Für Unternehmen, welche über keine oder nur begrenzte Erfahrung im Bereich CI/CD verfügen, ist von der Verwendung dieses Tools abzuraten. Stattdessen sollten diese das SAP CI/CD in Betracht ziehen, da dieses über konfigurierbare Templates verfügt, was bei der Implementierung der Pipelines eine deutlich geringere Expertise benötigt. Für Unternehmen, welche hingegen ein hohes Maß an Flexibilität erfordern, empfiehlt sich die Verwendung von Jenkins. Da dieses Tool in einem On-Premise-Modell betrieben wird, besitzen Unternehmen vollständige Kontrolle über das gesamte System. Dies ist insbesondere vorteilhaft für Unternehmen, welche sich in Branchen mit strikten Regularien befinden. Durch die Integration zahlreicher Plug-ins kann zum einen sichergestellt werden, dass alle in einem CI/CD-Prozess benötigten Compliance-Überprüfungen unterstützt werden. Darüber hinaus ermöglicht dies eine flexible Gestaltung der Systemsicherheit. In einer kritischen Reflexion stellt sich das AHP als ein geeignetes Verfahren zur Analyse von CI/CD-Tools dar. Mithilfe dieser Methode war es möglich, dass bei der Festlegung und Gewichtung der Bewertungskriterien, Präferenzen der verschiedenen an

der Bereitstellung von Software beteiligten Stakeholder berücksichtigt werden konnten. Dafür wurde ein Expertengremium aus Mitarbeitenden der SAP zusammengestellt, welche in verschiedensten Bereichen der Entwicklung und Bereitstellung von Software spezialisiert sind. Dadurch konnte ein umfassender Überblick über die Anforderungen des CI/CD-Prozesses erlangt werden. Die vorliegende Arbeit hat sich auf die Evaluation der CI/CD-Pipelines in Abhängigkeit der Technologien SAP CAP Node, SAP UI5 sowie Cloud Foundry beschränkt. Aufgrund der hohen Bedeutung der Technologieoffenheit, besteht jedoch die Möglichkeit, dass CEs divergierende Build-Tools, Test-Frameworks und Deploy- sowie Release-Funktionalitäten zur Implementierung der Pipelines benötigen. Dadurch könnte die in der vorliegenden Arbeit durchgeführte Bewertung divergierend ausfallen. Laut Experte 4, Test-Spezialist des SAP-internen CI/CD-Services, unterstützt Azure Pipelines im Vergleich zu anderen CI/CD-Lösungen eine Vielzahl von Technologien. Somit würde das Ergebnis der Bewertungen voraussichtlich ebenfalls bei der Berücksichtigung anderer Technologien ähnlich ausfallen. Folglich kann die im Rahmen dieser Arbeit durchgeführte Evaluation als angemessenes Ergebnis zur Automatisierung der Bereitstellungsprozesse einer CEA betrachtet werden [65, Z. 59 ff.].

6.2 Ausblick auf zukünftige Entwicklungen

Im Verlauf der Abhandlung hat sich gezeigt, dass die untersuchten CI/CD-Tools dazu geeignet sind, um Bereitstellungsprozesse zu automatisieren und die Qualität von IT-Services zu optimieren. Dabei besteht eine signifikante Wahrscheinlichkeit, dass CI/CD in naher Zukunft einer disruptiven Veränderung unterzogen wird. So gehen Forschungsinstitute wie die Linux Foundation davon aus, dass *Künstliche Intelligenz (KI)* zunehmend in die Bereitstellungsprozesse integriert wird [37]. Valerie Silverthorne, Executive Editor bei GitLab, bemerkt, dass diese Technologien das Potenzial besitzen, den Entwicklungsprozess zu revolutionieren und die Effizienz der Entwicklerteams zu erhöhen [53]. In diesem Kontext ergeben sich insbesondere drei Anwendungsfelder. Dazu gehören eine *Intelligente Fehlererkennung*, eine *Generative Testerstellung* sowie ein *KI-basiertes Monitoring*. Das Konzept der *Intelligenten*

Fehlererkennung stellt vermutlich den vielversprechendsten Bereich innerhalb des KI-gestützten CI/CDs dar. Durch dieses Verfahren ist es möglich, vorherzusagen, welche Code-Änderungen potenziell zu Problemen in den Produktionssystemen führen könnten. Das intelligente Generieren von Empfehlungen soll dabei zur Reduzierung des Arbeitsaufwands sowie zur Beschleunigung des Entwicklungsprozesses beitragen [53]. Ein weiteres KI-Anwendungsgebiet liegt in der *Generativen Testerstellung*. So ist durch den Einsatz intelligenter Algorithmen möglich, Testfälle zu generieren, um Schwachstellen im Code aufzudecken. Diese umfassen neben Unit-, Integration- sowie E2E-Tests ebenfalls intelligente Security-Analysen. Mit diesen soll während des gesamten Entwicklungsprozesses, also von der Programmierung über die Integration bis zur Bereitstellung, gewährleistet werden, dass Sicherheitslücken erkannt und potenzielle Risiken besser eingeschätzt werden [36]. Das automatische Generieren von Validierungs-Workflows sorgt darüber hinaus für eine höhere Testabdeckung und erlaubt den Entwicklern, sich verstärkt auf die Implementierung neuer Funktionalitäten zu fokussieren. Durch die im *KI-basierten Monitoring* abgewickelten Analysen kann der Bereitstellungs-Workflow optimal überwacht und der CI/CD-Prozess somit sukzessive optimiert werden. Im Zuge dessen kann dies ebenfalls dazu verwendet werden, eine optimale Bereitstellungsstrategie, wie das Blue-Green-Deployment, Canary-Deployment oder Shadow-Deployment, zu erörtern. Somit ist es Mithilfe von KI-Modellen möglich, den Erfolg der verschiedenen Bereitstellungskonzepte vorherzusagen und somit den Softwareauslieferungsprozess zu optimieren [36]. Unklar bleibt, wie sich diese Konzepte konkret in die Bereitstellungs-Workflows der CEs integrieren lassen und ob die in dieser Arbeit evaluierten CI/CD-Tools dafür geeignet sind. Eine weiterführende Fragestellung könnte daher wie folgt lauten:

Inwiefern lassen sich KI-basierte CI/CD-Konzepte in die Bereitstellungsprozesse der CEs integrieren und welches Tool birgt hierfür den größten Mehrwert?

Literatur

Print-Quellen

- [1] Matthias Biehl. *API architecture: The big picture for building APIs*. API-university series. API-University Press, 2015. ISBN: 9781508676645.
- [2] Ralf Bruns und Jürgen Dunkel. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Xpert.press. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 9783642024399. DOI: 10.1007/978-3-642-02439-9.
- [3] Lisa Ellram. „Total Cost of Ownership: Elements and Implementation“. In: *International Journal of Purchasing and Materials Management* 29.3 (1993), S. 2–11. ISSN: 10556001. DOI: 10.1111/j.1745-493X.1993.tb00013.x.
- [4] Kathleen Gerson und Sarah Damaske. *The science and art of interviewing*. New York, NY: Oxford University Press, 2021. ISBN: 9780199324293. DOI: 10.1093/oso/9780199324286.001.0001.
- [5] Joachim Goll und Daniel Hommel. *Mit Scrum zum gewünschten System*. Wiesbaden: Springer Vieweg, 2015. ISBN: 9783658107208. DOI: 10.1007/978-3-658-10721-5.
- [6] Jürgen Halstenberg. *DevOps: Ein Überblick*. Essentials Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020. 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*. 3rd ed. London, England: BCS Learning & Development Limited, 2015. 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*. 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*. Shelter Island, New York: Manning Publications Co, 2021. 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*. 2nd ed. Sebastopol, CA.: O'Reilly, 2012. 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] Panorama Consulting Solutions, Hrsg. *The 2019 ERP Report*. 2019. URL: <https://www.panorama-consulting.com/resource-center/erp-software-research-and-reports/panorama-consulting-solutions-2019-erp-report/> (besucht am 15.04.2023).
- [13] 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.
- [14] 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.
- [15] 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>.
- [16] Joseph T. Vesey. „Time-to-market: Put speed in product development“. In: *Industrial Marketing Management* 21.2 (1992), S. 151–158. ISSN: 0019-8501. DOI: 10.1016/0019-8501(92)90010-Q. URL: <https://www.sciencedirect.com/science/article/pii/001985019290010q>.

- [17] Wolfgang Vieweg. „Agiles (Projekt-)Management“. 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.
- [18] Alberto Vivencio, Hrsg. *Testmanagement Bei SAP-Projekten: Erfolgreich Planen * Steuern * Reporten Bei der Einführung Von SAP-Banking*. 1st ed. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2013. ISBN: 978-3-8348-1623-8. DOI: 10.1007/978-3-8348-2142-3.
- [19] Fiorella Zampetti u. a. „CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study“. In: *2021 IEEE International Conference 9/27/2021 - 2021*, S. 471–482. DOI: 10.1109/ICSME52107.2021.00048.

Online-Quellen

- [20] John Adam. *Was ist agile Softwareentwicklung?* Hrsg. von K&C. 2021. URL: <https://kruschecompany.com/de/agile-softwareentwicklung/> (besucht am 05.03.2023).
- [21] Shivam Arora. *What Is and What Are the Benefits of Docker Container?* Hrsg. von Simplilearn. 2023. URL: <https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container> (besucht am 09.04.2023).
- [22] Sourav Atta. *Monitor Jenkins Application Logs using ELK Stack - Sourav Atta*. 2020. URL: <https://souravatta.medium.com/monitor-jenkins-build-logs-using-elk-stack-697e13b78cb1> (besucht am 29.03.2023).
- [23] *Azure DevOps Services Pricing*. 2023. URL: <https://azure.microsoft.com/en-us/pricing/details/devops/devops-services/> (besucht am 10.04.2023).
- [24] *Azure Pipelines Extensions*. 2023. URL: <https://marketplace.visualstudio.com/search?target=AzureDevOps&category=Azure%20Pipelines&sortBy=Installs> (besucht am 10.04.2023).

- [25] Shweta Bhandal und Abby Taylor. *The Evolution from Agile to DevOps to Continuous Delivery* — Qentelli. Hrsg. von Qentelli. 2023. URL: <https://www.qentelli.com/thought-leadership/insights/evolution-agile-devops-continuous-delivery> (besucht am 05.03.2023).
- [26] Shreya Bose. *What is End To End Testing?* 2023. URL: <https://www.browserstack.com/guide/end-to-end-testing> (besucht am 08.03.2023).
- [27] Ian Buchannan. *Feature Flags*. 2023. URL: <https://www.atlassian.com/continuous-delivery/principles/feature-flags> (besucht am 09.04.2023).
- [28] *Build Stage - Project Piper: Continuous Delivery for the SAP Ecosystem*. 2023. URL: <https://www.project-piper.io/stages/build/> (besucht am 10.04.2023).
- [29] Anthony Cecchini. *What It Is and Why It's Important - Part 2*. 2021. URL: <https://itpfed.com/composable-erp-what-it-is-and-why-its-important-part-2/> (besucht am 18.04.2023).
- [30] *cloudFoundryDeploy - Project Piper: Continuous Delivery for the SAP Ecosystem*. 2023. URL: <https://www.project-piper.io/steps/cloudFoundryDeploy/#additional-hints> (besucht am 10.04.2023).
- [31] Codefresh, Hrsg. *CI/CD Pipelines for Microservices*. 2023. URL: <https://medium.com/containers-101/ci-cd-pipelines-for-microservices-ea33fb48dae0> (besucht am 02.04.2023).
- [32] Steve Danielson. *Konfigurieren von Pipelinetriggern - Azure Pipelines*. 2023. URL: <https://learn.microsoft.com/de-de/azure/devops/pipelines/process/pipeline-triggers?view=azure-devops> (besucht am 10.04.2023).
- [33] Steve Danielson. *Microsoft-hosted agents for Azure Pipelines - Azure Pipelines*. 2023. (Besucht am 10.04.2023).
- [34] 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).

- [35] Dora, Hrsg. *2022 State of DevOps Report - Google Cloud*. 2022. URL: <https://cloud.google.com/devops/state-of-devops> (besucht am 11.04.2023).
- [36] Dori Exterman. *DevOps and AI - A Match Made in Heaven*. 2022. URL: <https://www.incredibuild.com/blog/devops-and-ai-a-match-made-in-heaven> (besucht am 16.04.2023).
- [37] *Future of Continuous Delivery Trends - CD Foundation*. 2022. URL: <https://cd.foundation/blog/2022/04/01/future-of-continuous-delivery-trends/> (besucht am 16.04.2023).
- [38] Susan Galer. *Disrupted ERP Systems Rise to Reckoning in the Cloud*. 2022. URL: <https://news.sap.com/2022/10/composable-business-for-disrupted-erp-systems/> (besucht am 08.04.2023).
- [39] Dennis Gaughan u. a. *Future of Applications: Delivering the Composable Enterprise*. 2020. URL: <https://www.gartner.com/en/doc/465932-future-of-applications-delivering-the-composable-enterprise> (besucht am 18.04.2023).
- [40] *GitHub Authentication*. 2023. URL: <https://plugins.jenkins.io/github-oauth/> (besucht am 10.04.2023).
- [41] Srijani Gosh. *Top 10 Jenkins Plugins with Features for 2023*. 2023. URL: <https://www.knowledgehut.com/blog/devops/jenkins-plugins> (besucht am 10.04.2023).
- [42] 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).
- [43] 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).

- [44] *Multitarget Applications in the Cloud Foundry Environment*. 2023. URL: <https://help.sap.com/docs/btp/sap-business-technology-platform/multitarget-applications-in-cloud-foundry-environment> (besucht am 09.04.2023).
- [45] Kasey Panetta. *Top 10 Strategic Predictions for 2022 and Beyond*. 2021. URL: <https://www.gartner.com/en/articles/you-ll-be-breaking-up-with-bad-customers-and-9-other-predictions-for-2022-and-beyond> (besucht am 08.04.2023).
- [46] 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).
- [47] *Posts containing 'azure pipelines' - Stack Overflow*. 2023. URL: <https://stackoverflow.com/search?q=azure+Pipelines> (besucht am 03.04.2023).
- [48] *Posts containing 'jenkins' - Stack Overflow*. 2023. URL: <https://stackoverflow.com/search?q=jenkins> (besucht am 03.04.2023).
- [49] *Posts containing 'sap ci/cd' - Stack Overflow*. 2023. URL: <https://stackoverflow.com/search?q=SAP+ci%2FCD> (besucht am 03.04.2023).
- [50] *Release Stage - Project Piper: Continuous Delivery for the SAP Ecosystem*. 2023. URL: <https://www.project-piper.io/stages/release/> (besucht am 10.04.2023).
- [51] *Role-based Authorization Strategy*. 2023. URL: <https://plugins.jenkins.io/role-strategy/> (besucht am 10.04.2023).
- [52] Jörg Schönenstein. *Composable-Business und -Commerce durch MACH-Architektur*. Hrsg. von communicate AG. 2023. URL: <https://www.communicode.de/blog/work/composable-business-und-commerce-durch-mach-technologie> (besucht am 13.03.2023).

- [53] Valerie Silverthorne. *Why AI in DevOps is here to stay*. 2022. URL: <https://about.gitlab.com/blog/2022/09/15/why-ai-in-devops-is-here-to-stay/> (besucht am 16.04.2023).
- [54] Stack Overflow, Hrsg. *Stack Overflow Developer Survey 2020*. 2020. URL: <https://insights.stackoverflow.com/survey/2020> (besucht am 03.04.2023).
- [55] Stevens und Harald. *Interplay of SAP Cloud Platform Transport Management, CTS+ and ChaRM in hybrid landscapes*. 2023. URL: <https://blogs.sap.com/2020/01/31/interplay-of-sap-cloud-platform-transport-management-cts-and-charm-in-hybrid-landscapes/> (besucht am 27.03.2023).
- [56] Synopsys, Hrsg. *What Is CI/CD and How Does It Work?* 2023. URL: <https://www.synopsys.com/glossary/what-is-cicd.html> (besucht am 08.03.2023).
- [57] *The Cloud vs. On-Premise Cost: Which One is Cheaper?* 2023. URL: <https://www.executech.com/insights/the-cloud-vs-on-premise-cost-comparison/> (besucht am 10.04.2023).
- [58] *The Evolution of Container Usage at Netflix - Netflix TechBlog*. 2017. URL: <https://netflixtechblog.com/the-evolution-of-container-usage-at-netflix-3abfc096781b> (besucht am 10.04.2023).
- [59] 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).
- [60] *Using Jenkins agents*. 2023-04-10. URL: <https://www.jenkins.io/doc/book/using/using-agents/> (besucht am 10.04.2023).
- [61] *Vergleich der Azure-Supportpläne*. 2023. URL: <https://azure.microsoft.com/de-de/support/plans> (besucht am 10.04.2023).
- [62] *What Is SAP Continuous Integration and Delivery*. 2023. URL: https://help.sap.com/docs/CONTINUOUS_DELIVERY/99c72101f7ee40d0b2deb4df72ba1ad3/618ca03fdca24e56924cc87cfbb7673a.html?language=en-US (besucht am 09.04.2023).

Experteninterviews

- [63] Backend Test Developer SAP DTS Integration, Hrsg. *Expertengewichtung 4*. 29.03.2023.
- [64] Frontend Test Developer SAP Hyperspace Adoption & Onboarding, Hrsg. *Expertengewichtung 3*. 22.03.2023.
- [65] Frontend Test Developer SAP Hyperspace Adoption & Onboarding, Hrsg. *Experteninterview 4*. 22.03.2023.
- [66] Full-Stack-Entwickler SAP DTS Integration, Hrsg. *Expertengewichtung 2*. 21.03.2023.
- [67] Product Management CLM, Hrsg. *Expertengewichtung 5*. 29.03.2023.
- [68] Product Manager SAP Hyperspace CI/CD, Hrsg. *Experteninterview 2*. 24.03.2023.
- [69] Product Manager SAP Hyperspace Security Tools, Hrsg. *Experteninterview 3*. 22.03.2023.
- [70] Software Architekt SAP DTS Integration, Hrsg. *Expertengewichtung 1*. 27.03.2023.

Anhang

Anhangsverzeichnis

A	Allgemeine Ergänzungen	XIX
A.1	DevOps	XIX
A.2	Ramped-Deployment	XX
A.3	Feature Flags	XXI
B	Pipeline-Prototyp	XXII
B.1	Azure Pipelines	XXII
B.2	Jenkins	XXIX
B.3	SAP CI/CD	XXXII
C	Expertenmaterialien	XXXVII
C.1	Experteninterview 1	XXXVII
C.2	Experteninterview 2	XLII
C.3	Experteninterview 3	XLV
C.4	Experteninterview 4	XLVII
C.5	Kodierung der Experteninterviews	LI
C.6	Expertengewichtung 1	LXIII
C.7	Expertengewichtung 2	LXVIII
C.8	Expertengewichtung 3	LXXIII
C.9	Expertengewichtung 4	LXXVIII
C.10	Expertengewichtung 5	LXXXIII
C.11	Expertengewichtung Durchschnitt	LXXXVIII

Anhang A Allgemeine Ergänzungen

A.1 DevOps

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]. Der *Time-to-Market* beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [16, 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, emp-

fehlt angesichts der bei Softwareintegration entstehenden Komplexität, Anwendungen schnellst möglichst in produktionsähnlichen Umgebungen zu testen [16]. Aus diesen Erfahrungen sollen Best-Practises entwickelt werden, welche innerhalb von Teams und organisationsübergreifend weitergegeben werden (*Teilen*) [6, S. 7].

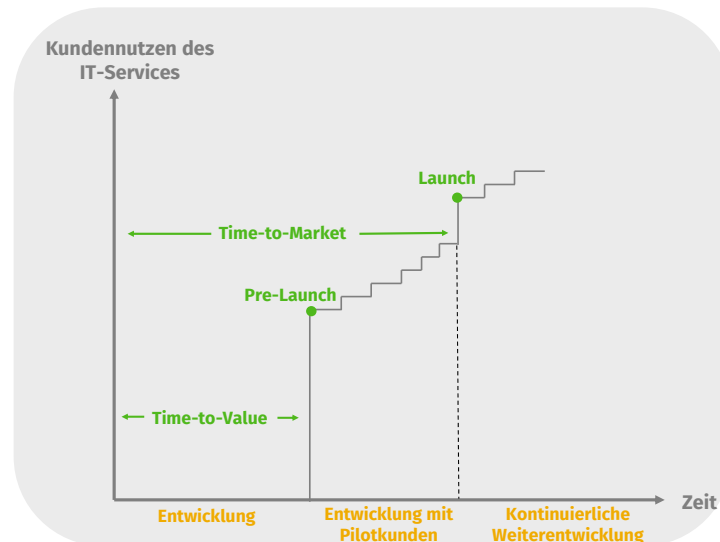


Abbildung 20: Zeitliche Darstellung der Herbeiführung von Kundennutzen bei der Entwicklung von IT-Services [6, S. 9].

A.2 Ramped-Deployment

Für Anwendungen auf einer kritischen IT-Infrastruktur wird i.d.R. die *Ramped-Deployment-Strategie* verwendet. Diese ermöglicht eine präzise Kontrolle horizontal skaliertter Services. Bei einer horizontalen Skalierung erfolgt die Replizierung identischer 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 [59].

A.3 Feature Flags

Bei dieser Methode wird die Sichtbarkeit neuer Funktionalitäten an eine *Flag* gekoppelt. Diese Flags repräsentieren globale Variablen, welche von dem Systemadministrator oder den Anwendern gesetzt werden können. Abhängig von dem Zustand der Flag kann gesteuert werden, ob die im CI/CD-Prozess bereitgestellten Features für Anwender sichtbar sind [27].

Anhang B Pipeline-Prototyp

B.1 Azure Pipelines

B.1.1 CI-Pipeline-Implementation

Die Pipeline-Implementation ist bei allen Services identisch:

```
trigger:
- main

jobs:
- job: Init
  pool:
    vmImage: ubuntu-latest
  steps:
- checkout: none
- task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
    cacheHitVar: FOUND_PIPER
    displayName: Cache piper go binary
- script: |
    mkdir -p bin
    curl -L --output bin/piper $(Piper_Lib_Url)
    chmod +x bin/piper
    condition: ne(variables.FOUND_PIPER, 'true')
    displayName: 'Download_Piper'
- script: bin/piper version
  displayName: 'Piper_Version'

- job: build
```

```

pool:
  vmImage: 'ubuntu-latest'
steps:
- task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
    displayName: deploy iflow
- script: |
  bin/piper npmExecuteScripts --runScripts=['
    initlint']
  displayName: 'Execute_npm_script'
- script: |
  bin/piper mtaBuild
  displayName: 'Execute_mbtBuild'

- job: Additional_Tests
  - task: Cache@2
    inputs:
      key: piper-go-official
      path: bin
      displayName: deploy iflow
  displayName: Additional_Tests
pool:
  vmImage: ubuntu-latest
steps:
- script: |
  bin/piper npmExecute --dockerImage=$(Docker-
    Image-Endpoint) --npmRunCommand='run_mykarma
    :ci'
  displayName: 'runKarmaTests'

```

B.1.2 CD-Pipeline-Implementation

Die Pipeline-Implementation ist bei allen Services identisch:

```
trigger:
- main

jobs:
- job: Init
  pool:
    vmImage: ubuntu-latest
  steps:
  - checkout: none
  - task: Cache@2
    inputs:
      key: piper-go-official
      path: bin
      cacheHitVar: FOUND_PIPER
      displayName: Cache piper go binary
  - script: |
      mkdir -p bin
      curl -L --output bin/piper $(
        Piper_Lib_Url)
      chmod +x bin/piper
      condition: ne(variables.FOUND_PIPER, '
        true')
      displayName: 'Download_Piper'
  - script: bin/piper version
    displayName: 'Piper_Version'

- job: Build
  pool:
```



```

        vmImage: 'ubuntu-latest'
steps:
- task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
    displayName: deploy iflow
- script: |
  bin/piper npmExecuteScripts --
    runScripts=['initlint']
    displayName: 'Execute_npm_script'
- script: |
  bin/piper mtaBuild
  displayName: 'Execute_mbtBuild'

- job: Additional_Tests
  task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
    displayName: deploy iflow
  displayName: Additional_Tests
  pool:
    vmImage: ubuntu-latest
  steps:
  - script: |
    bin/piper npmExecute --dockerImage=$(
      Docker-Image-Endpoint) --
      npmRunCommand='run_mykarma:ci'
      displayName: 'runKarmaTests'

- job: Acceptance

```

```

pool:
  vmImage: ubuntu-latest
steps:
- task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
    displayName: deploy iflow
- script: |
  bin/piper cloudFoundryDeploy
  displayName: 'Deploy'
- script: |
  bin/piper uiVeri5ExecuteTests
  displayName: 'Execute_WDI5'

- job: Compliance
  pool:
    vmImage: ubuntu-latest
  steps:
  - task: Cache@2
    inputs:
      key: piper-go-official
      path: bin
      displayName: deploy iflow
  - script: |
    bin/piper sonarExecuteScan --
      branchName='main' --githubToken=$(
        Github_Token) --token=$(Token) --
        serverUrl=$(serverUrl)
    displayName: 'sonarExecuteScan'

- job: Release




```

```

pool:
  vmImage: ubuntu-latest
steps:
- task: Cache@2
  inputs:
    key: piper-go-official
    path: bin
  displayName: deploy iflow piper
- script: |
  bin/piper cloudFoundryDeploy
  displayName: 'Deploy'

```

B.1.3 Performance-Ergebnisse

Jobs		
>  Init		13s
>  build		2m 15s
>  Additional_Tests		56s













Jobs		
Name	Status	Duration
 Init	Success	 15s
 build	Success	 2m 19s
 Additional_Tests	Success	 58s
 Acceptance	Success	 4m 39s
 Compliance	Success	 1m 28s
 Release	Success	 2m 32s

Abbildung 21: Integration- und Delivery-Zeit für den RiskService mit Azure Pipelines. Eigene Darstellung.

Jobs		
>	✓ Init	12s
>	✓ build	2m 12s
>	✓ Additional_Tests	58s

Jobs		
Name	Status	Duration
✓ Init	Success	🕒 14s
✓ build	Success	🕒 2m 13s
✓ Additional_Tests	Success	🕒 52s
✓ Acceptance	Success	🕒 4m 37s
✓ Compliance	Success	🕒 1m 27s
✓ Release	Success	🕒 2m 30s

Abbildung 22: Integration- und Delivery-Zeit für den SupplierService mit Azure Pipelines. Eigene Darstellung.

Jobs		
>	✓ Init	12s
>	✓ build	2m 10s
>	✓ Additional_Tests	52s

Jobs		
Name	Status	Duration
✓ Init	Success	🕒 15s
✓ Build	Success	🕒 2m 18s
✓ Additional_Tests	Success	🕒 59s
✓ Acceptance	Success	🕒 4m 46s
✓ Compliance	Success	🕒 1m 33s
✓ Release	Success	🕒 2m 39s

Abbildung 23: Integration- und Delivery-Zeit für den PlantService mit Azure Pipelines. Eigene Darstellung.

B.2 Jenkins

B.2.1 CI-Pipeline-Implementation

Die Pipeline-Implementation ist bei allen Services identisch:

```
@Library('piper-lib-os') _

node() {
    stage('Prepare') {
        deleteDir()
        checkout scm
        setupCommonPipelineEnvironment script:this
    }

    stage('Build') {
        npmExecuteScripts script: this, runScripts: ["initlint"]
        mtaBuild script:this
    }

    stage('Additional_Tests'){
        npmExecute script: this, dockerImage: (dockerImage),
        npmCommand : "run mykarma:ci"
    }
}
```

B.2.2 CD-Pipeline-Implementation

Die Pipeline-Implementation ist bei allen Services identisch:



```

@Library('piper-lib-os') _

node() {
    stage('Prepare') {
        deleteDir()
        checkout scm
        setupCommonPipelineEnvironment script:this
    }

    stage('Build')    {
        npmExecuteScripts script: this, runScripts: ["
            initlint"]
        mtaBuild script:this
    }

    stage('Additional_Tests'){
        npmExecute script: this, dockerImage: (dockerImage)
        , npmCommand : "run mykarma:ci"
    }

    stage('Acceptance') {
        cloudFoundryDeploy script: this
        uiVeri5ExecuteTests script: this
    }

    stage('Compliance') {
        sonarExecuteScan script:this, branchName: "main",
        githubToken: (githubToken), token:(Token),
        serverUrl: (serverUrl)
    }

    stage('Deploy')    {

```

```

cloudFoundryDeploy script:this
}
}

```

B.2.3 Performance-Ergebnisse

Stage View

	Prepare	Build	Additional_Tests	Acceptance	Compliance	Deploy
#104 Apr 07 13:17 2 commits	5s	4min 05s	2min 16s	7min 48s	1min 51s	2min 59s
#103 Apr 07 12:40 2 commits	6s	4min 13s	2min 08s			

Abbildung 24: Integration- und Delivery-Zeit für den RiskService mit Jenkins. Eigene Darstellung.

Stage View

	Prepare	Build	Additional_Tests	Acceptance	Compliance	Deploy
#99 Apr 07 11:46 2 commits	5s	3min 58s	2min 5s	7min 35s	1min 46s	2min 53s
#98 Apr 07 11:10 6 commits	6s	4min 3s	1min 55s			

Abbildung 25: Integration- und Delivery-Zeit für den SupplierService mit Jenkins. Eigene Darstellung.

Stage View

	Prepare	Build	Additional_Tests	Acceptance	Compliance	Deploy
#111 Apr 07 14:52 2 commits	6s	4min 06s	2min 14s	7min 53s	1min 53s	2min 58s
#110 Apr 07 14:16 2 commits	6s	4min 7s	2min 10s			

Abbildung 26: Integration- und Delivery-Zeit für den PlantService mit Jenkins. Eigene Darstellung.

B.3 SAP CI/CD



B.3.1 CI-Pipeline-Konfiguration



Abbildung 27: CI-Pipeline-Konfiguration für SAP CI/CD

B.3.2 CD-Pipeline-Konfiguration


STAGES


Configuration Mode: Job Editor   YML

Build


Build Tool: mta

Build Tool Version: Java 11 Node 16


Maven Static Code Checks 

Lint Check 

☐ Fail on Error

Additional Unit Tests 

npm Script: mykarma

Acceptance 

Deploy to Cloud Foundry Space


API Endpoint: <https://api.cf.eu10.hana.ondemand.com>

Org Name: cf-dts-integration-de

Space: i538951_dev

Deploy Type: standard


Credentials: cf-creds-i538951

WebdriverIO Test 

npm Script: uiVeri5

Base URL: https://cf-dts-integration-de.launchpad.cfapps.eu10.hana.ondemand.com/i538951_service.nssuppliermanagement-1.0.0/index.html#fe-trop-v4

Credentials: cf-creds-i538951

SonarQube Scan 


Mode: SonarCloud

URL: <https://sonar.tools.sap/>

Organization: sonar.tools.sap

Project Key: i538951_ba

SonarQube Token Credentials: i538951-sonar-creds

Release 

Deploy to Cloud Foundry Space


API Endpoint: <https://api.cf.eu10.hana.ondemand.com>

Org Name: cf-dts-integration-de

Space: i538951_dev

Deploy Type: standard

Credentials: cf-creds-i538951

Upload to Cloud Transport Management 

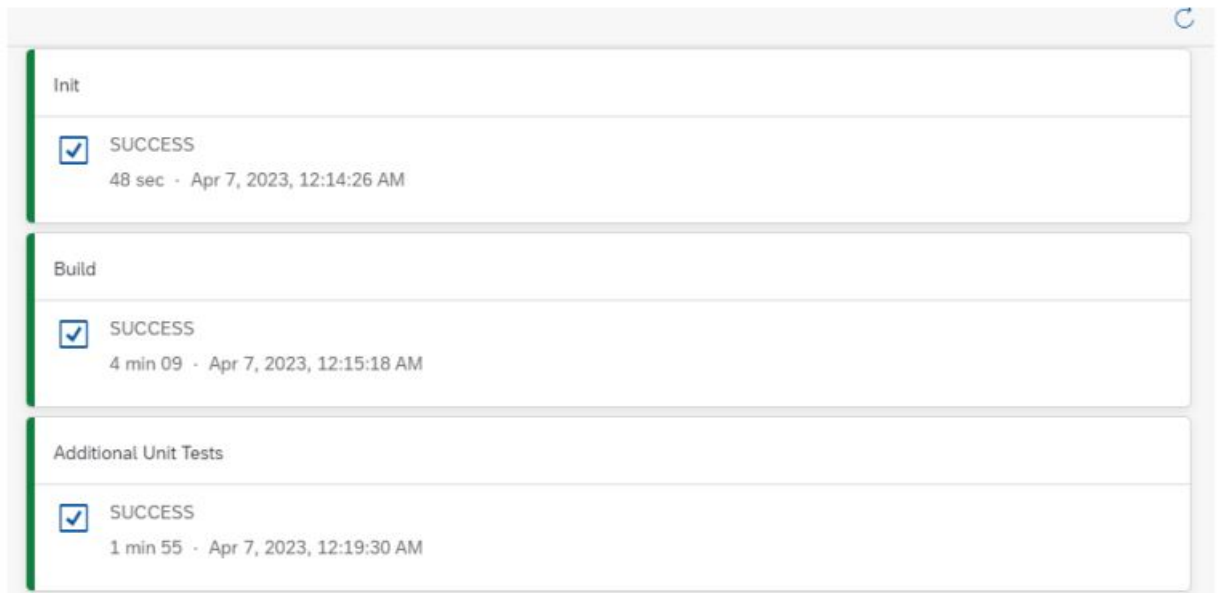
Node Name:

Service Key:

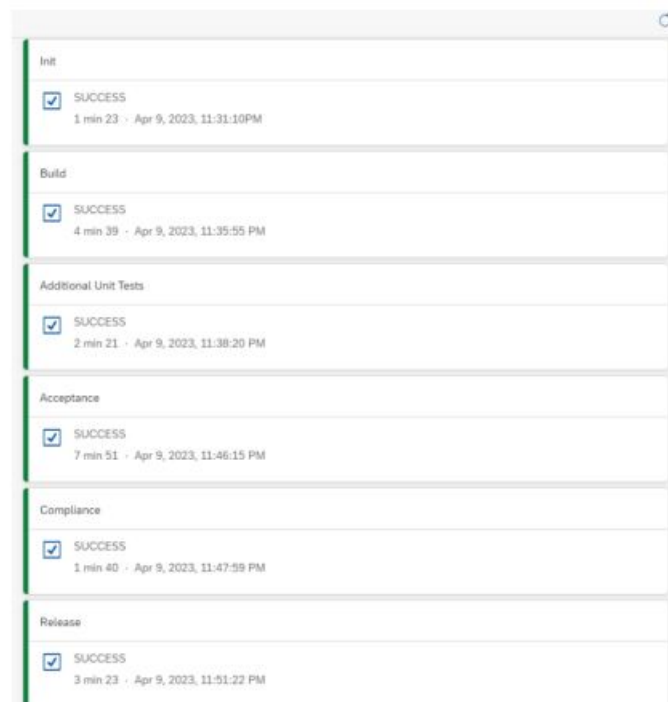
Abbildung 28: CD-Pipeline-Konfiguration für SAP CI/CD

XXXIII

B.3.3 Performance-Ergebnisse



Init
<input checked="" type="checkbox"/> SUCCESS 48 sec · Apr 7, 2023, 12:14:26 AM
Build
<input checked="" type="checkbox"/> SUCCESS 4 min 09 · Apr 7, 2023, 12:15:18 AM
Additional Unit Tests
<input checked="" type="checkbox"/> SUCCESS 1 min 55 · Apr 7, 2023, 12:19:30 AM



Init
<input checked="" type="checkbox"/> SUCCESS 1 min 23 · Apr 9, 2023, 11:31:10PM
Build
<input checked="" type="checkbox"/> SUCCESS 4 min 39 · Apr 9, 2023, 11:35:55 PM
Additional Unit Tests
<input checked="" type="checkbox"/> SUCCESS 2 min 21 · Apr 9, 2023, 11:38:20 PM
Acceptance
<input checked="" type="checkbox"/> SUCCESS 7 min 51 · Apr 9, 2023, 11:46:15 PM
Compliance
<input checked="" type="checkbox"/> SUCCESS 1 min 40 · Apr 9, 2023, 11:47:59 PM
Release
<input checked="" type="checkbox"/> SUCCESS 3 min 23 · Apr 9, 2023, 11:51:22 PM

Abbildung 29: Integration- und Delivery-Zeit für den RiskService mit SAP CI/CD. Eigene Darstellung.

Init
<input checked="" type="checkbox"/> SUCCESS 48 sec - Apr 7, 2023, 11:38:26 AM
Build
<input checked="" type="checkbox"/> SUCCESS 4 min 07 - Apr 7, 2023, 11:39:15 AM
Additional Unit Tests
<input checked="" type="checkbox"/> SUCCESS 1 min 59 - Apr 7, 2023, 11:43:23 AM

Init
<input checked="" type="checkbox"/> SUCCESS 1 min 19 - Apr 9, 2023, 11:04:00 PM
Build
<input checked="" type="checkbox"/> SUCCESS 4 min 32 - Apr 9, 2023, 11:05:20 PM
Additional Unit Tests
<input checked="" type="checkbox"/> SUCCESS 2 min 15 - Apr 9, 2023, 11:09:52 PM
Acceptance
<input checked="" type="checkbox"/> SUCCESS 7 min 55 - Apr 9, 2023, 11:12:08 PM
Compliance
<input checked="" type="checkbox"/> SUCCESS 1 min 29 - Apr 9, 2023, 11:20:04 PM
Release
<input checked="" type="checkbox"/> SUCCESS 3 min 18 - Apr 9, 2023, 11:21:33 PM
Declarative: Post Actions
<input checked="" type="checkbox"/> SUCCESS 90 ms - Apr 9, 2023, 11:24:52 PM

Abbildung 30: Integration- und Delivery-Zeit für den SupplierService mit SAP CI/CD. Eigene Darstellung.

Init	
<input checked="" type="checkbox"/>	SUCCESS 47 sec · Apr 7, 2023, 12:32:02 AM
Build	
<input checked="" type="checkbox"/>	SUCCESS 4 min 11 · Apr 7, 2023, 12:32:51 AM
Additional Unit Tests	
<input checked="" type="checkbox"/>	SUCCESS 1 min 59 · Apr 7, 2023, 12:37:15 AM

Init	
<input checked="" type="checkbox"/>	SUCCESS 1 min 21 · Apr 10, 2023, 06:04:12 PM
Build	
<input checked="" type="checkbox"/>	SUCCESS 4 min 42 · Apr 10, 2023, 06:05:35 PM
Additional Unit Tests	
<input checked="" type="checkbox"/>	SUCCESS 2 min 23 · Apr 10, 2023, 06:10:19 PM
Acceptance	
<input checked="" type="checkbox"/>	SUCCESS 7 min 48 · Apr 10, 2023, 06:12:45 PM
Compliance	
<input checked="" type="checkbox"/>	SUCCESS 1 min 43 · Apr 10, 2023, 06:20:40 PM
Release	
<input checked="" type="checkbox"/>	SUCCESS 3 min 29 · Apr 10, 2023, 06:22:25 PM

Abbildung 31: Integration- und Delivery-Zeit für den PlantService mit SAP CI/CD. Eigene Darstellung.

Anhang C Expertenmaterialien

C.1 Experteninterview 1

Interviewpartner: Product Owner SAP BTP Prod&Infra (Experte 1)

Datum: 17.03.2023

Interview-Medium: Microsoft-Teams

1 **Interviewer:** Du kannst dich ja mal kurz vorstellen und erläutern, was du be-
2 reits mit dem CI/CD-Bereich zu tun hattest und was deine täglichen Aufgaben sind.

3 **Experte:** Ich bin Product Owner für den Continuous Integration and Delivery Ser-
4 vice. Meine tägliche Aufgabe ist die Steuerung des Backlogs für unsere Anforderun-
5 gen. Dabei muss ich die Anforderungen, die über verschiedene Kanäle von unseren
6 Kunden hereinkommen konsolidieren und für unsere Abteilung bereitstellen.

7 **Interviewer:** Wie definierst du den Begriff CI/CD?

8 **Experte:** Also für mich gibt es einmal den CI Begriff. Dabei habe ich einen CI-
9 Server, der mir nach einem Push in mein zentrales Repository innerhalb kurzer Zeit
10 ein Feedback gibt. Danach kommt der CD-Prozess. Dabei kann ich abhängig von
11 verschiedenen Mechanismen, wie zum Beispiel einem Review oder einem Request die
12 CD-Pipeline auslösen. Die ist i.d.R. auch mächtiger als die CI-Pipeline. Mit dieser
13 wird zentral gebaut, getestet und ggf. auch noch Sachen wie Compliance, Vulnerabi-
14 lities, statische Codechecks, Integrations-Tests und Performance-Tests abgewickelt.
15 Das getestete Programm kann dann anschließend z.B. in ein Artefakt-Repository
16 oder in eine Produktionsumgebung bereitgestellt werden.

17 **Interviewer:** Welche Vorteile hat es, wenn Software kontinuierlich bereitgestellt
18 wird?

19 **Experte:** Der Vorteil ist der, dass ich meine Änderungen in kleinen Paketen, die
20 sich auch leichter integrieren lassen, vollziehe. Wenn ich tägliche oder alle zwei bis
21 drei Tage Änderungen mache und dann jeweils schaue, ob der Status noch grün ist,
22 birgt das gegenüber dem klassischen Wasserfallmodell sehr viele Vorteile. So kann
23 ich, wenn ich schnell in eine Canary-Umgebung bereitstelle, natürlich auch früher

24 Fehler finden, was dann im Endeffekt auch deutlich günstiger wird.

25 **Interviewer:** Welche unterschiedlichen Arten von Pipelines gibt es?

26 **Experte:** Das hängt ein wenig von den Anforderungen ab. Also typischerweise hat
27 man eine sehr kleine Pipeline für Request-Votes. Diese wird automatisch ausgelöst,
28 wenn in dem Github ein Pull-Request aufgemacht wird. Diese sollte maximal 10
29 bis 15 Minuten Laufzeit besitzen. So soll der Entwickler ein schnelles Feedback be-
30 kommen. Dann gibt es noch die Delivery-Pipelines. Diese wird dazu verwendet, um
31 in ein Artefakt-Repository oder auf die Produktionsumgebung bereitzustellen. Für
32 solche Pipelines kann entschieden werden, ob entweder alles am Stück gemacht wird
33 oder ob Komponenten aufgeteilt werden. Es bietet sich z.B. an, dass zu Beginn ein-
34 fache Unit- und Code-Tests gemacht werden und das Artefakt anschließend in das
35 Repository bereitgestellt wird. Konkurrent kann ebenfalls ein Job eingestellt wer-
36 den, welcher zu einem bestimmten Zeitpunkt durchläuft und ebenfalls aufwändigere
37 Tests abwickelt.

38 **Interviewer:** Du hattest Artefakt-Repository genannt. Welche Vorteile hat das
39 Artefakt-Repository?

40 **Experte:** Das spielt insbesondere bei einer CEA eine wichtige Rolle. Kleine ent-
41 wickelte Komponenten können mit Versionierung in das Artefakt-Repository be-
42 reitgestellt werden. Andere Entwickler können diese Komponente dann aus dem
43 Artefakt-Repository herausziehen und für eigenen Entwicklungen wiederverwenden.

44 **Interviewer:** Welche Stages hat eine typische CI/CD-Pipeline?

45 **Experte:** Typischerweise beginnt eine CI/CD-Pipeline mit dem Build-Stage bei wel-
46 cher Unit-Tests ausgeführt werden. Für CAP werden dabei die Frameworks Mocha
47 oder Jest verwendet. Dann gibt es eine Acceptance-Stage in welcher Akzeptanztests
48 ausgeführt werden. Solche Akzeptanztests können dann z.B. Integration-Tests um-
49 fassen, welche bei CAP-Node-Anwendungen mit Newmann automatisiert werden.
50 Dann gibt es eine Compliance-Stage. In dieser laufen Tools wie die SonarQube.
51 Dort wird z.B. geprüft, ob irgendwelche Lizenzrechte im Code verletzt werden.
52 Anschließend kommt die Security-Stage. In dieser wird nach Vulnerabilities und
53 statischen Kontexten geprüft und evaluiert, ob Gefahr für Cross-Skripting, Null-

54 Pointer-Exceptions etc. besteht. Dann gibt es noch die Release-Stage. Dort wird die
55 Anwendung dann tatsächlich auf die Cloud-Plattform bereitgestellt.

56 **Interviewer:** Welche Pipelines werden bei der SAP verwendet?

57 **Experte:** Zum einen wird der von der SAP bereitgestellte CI/CD-Service verwen-
58 det. Dieser sollte aber lediglich von Kunden verwendet werden. Dieser eignet sich
59 insbesondere bei weniger komplexen Anwendungen. Des Weiteren gibt es Jenkins.
60 Diese wird i.d.R. mit Project Piper verwendet. Die Jenkins-Pipeline muss dabei
61 selbst gehostet werden. Für interne Projekte wird dafür das Jenkins-as-a-Service
62 angeboten. Häufig wird für interne Projekte ebenfalls Azure Pipelines verwendet.

63 **Interviewer:** Welche Aspekte sind bei der Wahl eines CI/CD-Pipeline-Tools zu be-
64 achten?

65 **Experte:** Zum einen wie viel Wissen ein Team bereits im DevOps besitzt. Da-
66 bei sollte evaluiert werden, ob das Team bereits DevOps-Spezialisten besitzt, welche
67 schon häufig Pipelines implementiert haben. Für Abteilungen, welche keine DevOps-
68 Spezialisten haben, spielt die Benutzerfreundlichkeit eine große Rolle. Da ist es zum
69 einen wichtig, wie leicht sich die Tools warten lassen, aber auch, wie leicht sich ei-
70 ne Pipeline implementieren lässt. Weiterhin ist wichtig zu wissen, wie flexibel man
71 bei der Pipeline-Gestaltung sein will. Zudem muss natürlich auch evaluiert wer-
72 den, welche Funktionalitäten, also Tests, Code-Scans und Builds auf der Pipeline
73 ausgeführt werden sollen. In Bezug auf die Funktionalität sollte ebenfalls evaluiert
74 werden, auf welcher Plattform die Software bereitgestellt werden soll. Insbesondere
75 für CEA spielt ebenfalls die Skalierbarkeit eine wichtige Rolle. Horizontale Skalier-
76 barkeit umschreibt in diesem Kontext, dass mehrere Build gleichzeitig durchgeführt
77 werden können. Die vertikale Skalierbarkeit bedeutet, dass die Ressourcen einer
78 Pipeline-Instanz flexibel angepasst werden können. Da Jenkins selbst gehostet wird,
79 hat das natürlich für die vertikale Skalierbarkeit einen erheblichen Nachteil. Für vie-
80 le Kunden spielt ebenfalls die Sicherheit eine ausschlaggebende Rolle. Ein sicheres
81 System ist essenziell, damit in die Produktionsumgebungen keine Maleware einge-
82 schleust werden kann.

83 **Interviewer:** Wie sieht es mit der Unterstützung von Tests aus?

84 **Experte:** Es ist eigentlich fast alles auf dem SAP BTP CI/CD-Service möglich.
85 Was bisher noch nicht wirklich unterstützt wird, sind API-Tests.

86 **Interviewer:** Wie sieht es mit den Integrationsmöglichkeiten aus. Worauf muss da-
87 bei geachtet werden?

88 **Experte:** Die Integration ist ebenfalls ein sehr wichtiger Aspekt bei der Auswahl ei-
89 ner CI/CD-Pipeline. Dabei muss darauf geachtet werden, dass die Pipeline mit dem
90 Repository integrierbar ist. Der SAP CI/CD-Service unterstützt einen ganz nor-
91 malen Git-Server. Was ebenfalls funktioniert, sind BitBucket Repositorys. Für die
92 Integration wird dabei jedoch eine Webhook benötigt. Hierbei können jedoch aus-
93 schließlich Commit Events verarbeitet werden. Sehr selten wird eine CI/CD-Pipeline
94 auch in die Entwicklungsumgebung integriert. Das ist mit dem SAP CI/CD-Service
95 nicht möglich. SAP CI/CD hat diese Möglichkeit nicht. Jenkins und Azure können
96 dabei sowohl in Eclipse als auch VSCode eingebunden werden.

97 **Interviewer:** Gibt es irgendwelche Einschränkungen bezüglich der Laufzeitumge-
98 bung?

99 **Experte:** Bei unserem Service nicht. Wir können sowohl auf Cloud Foundry als
100 auch auf Kyma deployen.

101 **Interviewer:** Gibt es in dem SAP CI/CD-Tool irgendwelche Überwachungsfun-
102 ktionalitäten?

103 **Experte:** Für das SAP CI/CD können ausschließlich Pipeline-Logs ausgegeben wer-
104 den. Da gibt es verschiedene Notification-Services, die über den Erfolg der Pipeline-
105 Builds benachrichtigen. Aber ein direktes Monitoring der Pipeline gibt es nicht.

106 **Interviewer:** Welche Kosten fallen für die CI/CD-Pipeline an?

107 **Experte:** Eine Build-Hour kostet einen Euro.

108 **Interviewer:** Sind parallele Builds möglich und ist die Pipeline mit dem Transport
109 Management System integrierbar?

110 **Experte:** Nein leider nicht. Aber die Pipeline kann Software auf das Transport Ma-
111 nagement System bereitstellen.

112 **Interviewer:** Welche Support-Möglichkeiten stehen für SAP CI/CD bereit?

113 **Experte:** Wir bieten wie andere SAP-Cloud-Dienste ein Ticket-System mit Service

114 Now an. Wenn Kunden konkrete technische Unterstützung benötigen, können sich
115 diese an die technischen Berater der SAP wenden.

116

C.2 Experteninterview 2

Interviewpartner: Product Manager SAP Hyperspace CI/CD (Experte 2)

1 Datum: 24.03.2023

2 Interview-Medium: Microsoft-Teams

3 **Interviewer:** Du kannst dich nun gerne vorstellen. Wie kommst du während deiner
4 Arbeit mit CI/CD in Berührung?

5 **Experte:** Ich bin Product Manager. Ich habe zuerst für den SAP CI/CD-Service
6 gearbeitet. Nun bin ich im Hyperspace.

7 **Interviewer:** Was bedeutet für dich CI/CD?

8 **Experte:** CI ist die Integration, bei welchem die Änderungen von unterschiedlichen
9 Entwicklern so schnell wie möglich ein zentrales Repository integriert werden. CD
10 ist Continuous Delivery. Das ist die Möglichkeit, ein Feature so schnell wie möglich
11 auf die Produktion zu überführen und für den Kunden bereitzustellen.

12 **Interviewer:** Welchen Vorteil hat es Software schnell bereitzustellen?

13 **Experte:** Der Vorteil für mich ist, dass man mit kleineren Paketen arbeitet. So
14 ist die Gefahr, dass etwas im Produktivsystem kaputtgeht sehr gering. Mit klei-
15 nen Änderungen sind die Auswirkungen, die eine Integration hat, auch besser zu
16 überblicken.

17 **Interviewer:** Aus welchen typischen Komponenten besteht eine gewöhnliche Pipe-
18 line?

19 **Experte:** Also man fängt typischerweise mit dem Sync auf seinem Git Repository
20 an. Der zweite Schritt ist dann der Build. Dort werden auch Unit-, Integration- und
21 Acceptance-Tests in unterschiedlichen Spaces ausgeführt. In einer Acceptance-Stage
22 werden dann mehr Tests als in der Build-Phase durchgeführt. Dazu gehören neben
23 normalen Tests auch Security-Scans. Zuletzt wird die Software bereitgestellt.

24 **Interviewer:** Wie sind Pipelines aufgebaut?

25 **Experte:** Also früher haben wir keine unterschiedlichen Pipelines für CI und CD
26 verwendet. Da haben wir aber gesehen, dass das ziemlich problematisch ist. Wenn
27 alles sequenziell ausgeführt wird und dann in der Mitte irgendwas abbricht, ist mit

28 diesem Vorgehen sehr viel Zeit verloren gegangen. Nun geht der Trend in Richtung
29 Shift-Left. Die Pipelines werden somit deutlich verkleinert, womit schnelleres Feed-
30 back gegeben werden kann.

31 **Interviewer:** Welche Kriterien sollten bei der Auswahl von Pipelines beachtet wer-
32 den.

33 **Experte:** Es kommt natürlich darauf an, welche Produktstandards vorgegeben sind.
34 Wenn die Standards hoch sind, ist auch die Test-Funktionalität sehr wichtig. Dazu
35 gehören insbesondere Security-Checks, wie mit Fortify. Für sehr großen Entwick-
36 lungsprojekte ist es ebenfalls essenziell, dass die Pipelines eine gute Performance
37 besitzen. Somit kann Software schneller bereitgestellt werden. Des Weiteren ist es
38 essenziell, dass die CI/CD-Prozesse überwacht werden können. Gerade bei komple-
39 xen Systemen mit vielen Services ist das für den DevOps-Engineer oft die einzige
40 Möglichkeit, die Prozesse nachhaltig zu überblicken. Dann ist natürlich auch wichtig,
41 wie gut sich die Pipeline in die Infrastruktur integrieren lässt. Bei dieser Integra-
42 tion sollten jedoch jegliche Sicherheitsstandards eingehalten werden. Insbesondere
43 für kleinere Kunden ist es essenziell, welche Kosten durch die Pipeline verursacht
44 werden. Für viele Kunden ist darüber hinaus der Support wichtig. Es sollten konti-
45 nuierliche Updates, Schulungsmaterial sowie wie eine gute Dokumentation verfügbar
46 sein. Außerdem ist es für Entwickler immer vorteilhaft, wenn für die Tools eine große
47 Community existiert.

48 **Interviewer:** Mit welchen Pipelines hattest du bisher Erfahrung?

49 **Experte:** Mit Azure DevOps habe ich bisher noch nicht so viel Erfahrung gemacht.
50 In meinem jetzigen Team arbeite ich mit dem Jenkins-as-a-Service. In meinem vorhe-
51 rigen Team habe ich Erfahrung mit dem SAP CI/CD-Service gemacht. Ursprünglich
52 wurde das Projekt Piper für Jenkins nur für interne Projekte genutzt. Mittlerweile
53 wurde die Bibliothek als Open-Source veröffentlicht. Für interne Projekte darf das
54 SAP CI/CD aufgrund der derzeitigen Produktstandards nicht verwendet werden.
55 Dieser wird eigentlich nur für Kunden angeboten. Das SAP CI/CD-Tool lohnt sich
56 insbesondere für Kunden, welche noch nicht viel DevOps-Expertise besitzen und
57 auch keine teure Infrastruktur betreiben wollen.

58 **Interviewer:** Ist in dem CI/CD-Service von der SAP schon der Preis für Tools wie
59 SonarQube einberechnet?

60 **Experte:** Nein, der Preis ist nicht einberechnet. Tools wie SonarQube müssen von
61 den Kunden selbst gehostet werden.

62 **Interviewer:** Weißt du, ob die Tools in das SAP CTM integrierbar sind?

63 **Experte:** Ja, sowohl JaaS als auch SAP BTP CI/CD sind in das SAP CTM inte-
64 grierbar. Intern habe ich noch nicht oft gehört, dass dieser verwendet wird. Aber
65 Kunden können theoretisch auf das SAP CTM bereitstellen. Das SAP CTM ist
66 ebenfalls mit einem Change Management Surface verbunden. Damit kann man einen
67 Change-Auftrag erstellen und Artefakte zwischen unterschiedlichen Systemen bereit-
68 stellen. Dadurch hat man einfach mehr Transparenz. Außerdem bietet das System
69 für Composable-ERP-Systeme einen großen Vorteil. Über das SAP CTM können
70 Abhängigkeiten zwischen verschiedenen Microservices definiert werden.

71 **Interviewer:** Welche Überwachungsfunktionalitäten bieten die Pipelines und wel-
72 che Tools werden von Kunden i.d.R. verwendet?

73 **Experte:** Bei Jenkins weiß ich, dass man Logs auslesen und den Workflow so-
74 mit nachvollziehen kann. Für Monitoring wird innerhalb der SAP i.d.R. das SAP-
75 Partner-Tool Splunk verwendet. Das ist über Project Piper einbindbar. Kunden
76 nutzen häufig auch andere Open-Source-Tools. Ein sehr häufig verwendetes Tool ist
77 dabei das Kibana-Dashboard.

C.3 Experteninterview 3

Interviewpartner: Product Manager SAP Hyperspace Security Tools (Experte 3)

Datum: 22.03.2023

Interview-Medium: Microsoft-Teams

1 **Interviewer:** Wie hat sich das Thema Security im CI/CD-Kontext verändert?

2 **Experte:** In letzter Zeit hat sich das Thema Shift Left sehr stark etabliert. Das be-
3 deutet, dass das Feedback eigentlich möglichst früh an den Entwickler zurückgegeben
4 wird. Der Entwickler lernt somit viel nachhaltiger, da er im Integration-Kontext noch
5 keine große Verantwortung hat und somit kleine Arbeitspakete zur Verfügung ge-
6 stellt bekommt. Falls hingegen große Pakete in die Main-Line integriert werden, ist
7 irgendwann nicht mehr ersichtlich, wer welche Änderungen gemacht hat. Früher gab
8 es dabei immer einen Security-Experten, welcher sich vor der Auslieferung um alles
9 kümmern musste.

10 **Interviewer:** Wie wird Security in DevOps heutzutage gemacht?

11 **Experte:** Security sollte nicht mehr nur von einem Spezialisten behandelt werden.
12 Vielmehr sollte dies als Kollektiv vorangetrieben werden. Jeder muss bei der Ent-
13 wicklung seiner Funktionalitäten schon so früh wie möglich schauen, ob alle sicher-
14 heitsrelevanten Aspekte eingehalten wurden. Das wird dann i.d.R. durch Automati-
15 sierung gemacht. Es wird dabei schon sehr lange mit Security-Tools gearbeitet. His-
16 torische Tools sind dabei aber nicht sehr benutzerfreundlich. D.h., dass die Findings
17 nicht gut präsentiert werden. Somit versteht ein normaler Entwickler nicht, was mit
18 einem Finding gemeint ist und wie dieses Problem behoben werden kann. Da haben
19 sich in der letzten Zeit aber sehr viele neue und benutzerfreundlichere Tools etabliert.
20 Diese sollen bei der Realisierung des Shift-Left-Ansatzes unterstützen. Gerade bei
21 der Bereitstellung von ERP-Funktionalitäten sind Security-Scans sehr wichtig. Bei
22 der SAP gelten hierfür sehr strikte Produktstandards. Diese Security-Scans können
23 dabei sehr komplex sein. So ist es die Regel, dass ein Testdurchlauf auch mal mehr
24 als fünf Stunden Zeit in Anspruch nimmt.

25 **Interviewer:** Welche Arten von Security-Tools gibt es?

26 **Experte:** Es gibt i.d.R. zwei verschiedene Arten von Security-Tools. Es gibt da zum
27 einen die statischen Code-Analysen (SAST). Dort wird insbesondere OS-Scanning
28 betrieben. Dann gibt es noch das Dynamic Application Security Testing (DAST).
29 Dort werden dann auch tatsächlich UI-Elemente, APIs sowie Datenbanken gescannt.
30 Damit können im Produktionssystem leichter Scripting-Attacken oder SQL-Injections
31 verhindert werden. Manchmal wird dann auch noch die Kategorie des Interactive
32 Application Security Testing (IAST) definiert. Dabei wird ein Agent in die Laufzeit-
33 umgebung integriert, welcher die Insights der Analysen liefert. Dort können dann
34 z.B. Software-Component-Analysen gemacht werden, bei welchen HTTP-Requests
35 gespooft werden.

36 **Interviewer:** Welche Tools werden bei der SAP mit Project Piper verwendet?

37 **Experte:** Zum einen gibt es Fortify. Das ist insbesondere für Java und Python. Das
38 Tool ist allerdings kaum noch in Verwendung, da es sich historisch nicht weiterentwi-
39 ckelt hat. In näherer Zukunft wird dieses durch GitHub Advanced Security abgelöst.
40 Für CAP Node wird in den Produktstandards das Tool Checkmarx vorgeschrieben.
41 Für Open-Source ist das Tool Whitesource vorgeschrieben. Für SAP UI5 wird bei
42 der statischen Code-Analyse Checkmarx verwendet. Für Open-Source gibt es keine
43 Vorgabe. Das liegt daran, dass UI5 eigentlich über JavaScript geschrieben wird und
44 somit NPM als Package-Manager benötigt. Node wird in dieser Technologie jedoch
45 nicht unterstützt. Für statische Codeanalysen wird SonarQube und Lint verwendet.

C.4 Experteninterview 4

Interviewpartner: Frontend-Test-Entwickler SAP Hyperspace (Experte 4)

Datum: 22.03.2023

Interview-Medium: Microsoft-Teams

1 **Interviewer:** Du kannst dich nun gerne vorstellen.

2 **Experte:** Derzeit bin ich im Adoption and Onboarding Team von Hyperspace. Wir
3 unterstützen Kunden darin, ihre Projekte zu onboarden. Dafür bieten wird verschie-
4 dene Toolings, wie Security-Tools, Deployment-Tools, Test-Tools etc. an.

5 **Interviewer:** Was hat das Hyperspace mit CI/CD zu tun?

6 **Experte:** Hyperspace ist eine Plattform, die es ermöglicht, das CI/CD-Set-up mög-
7 lichst konsistent und einfach aufzusetzen. Somit soll der kognitive Load in den Teams
8 reduziert werden, um nicht alles manuell aufsetzen zu müssen. Gerade das Aufset-
9 zen einer Pipeline mit Jenkins, bei welchem u.a. auch Groovy-Scripten usw. benötigt
10 werden, kann einen sehr hohen Aufwand darstellen. Zudem benötigt dies sehr viel
11 Wissen. Hyperspace gibt den Entwicklungsteams Guidelines vor. D.h. auf Hyper-
12 space kann ich einfach ein Template auswählen. Das Hyperspace kümmert sich dann
13 darum, dass alle benötigten Tools zur Verfügung stehen und nicht alles selbst kon-
14 figuriert werden muss.

15 **Interviewer:** Wird im Hyperspace auch eine konkrete Step-Implementierung abge-
16 nommen?

17 **Experte:** Hyperspace erzeugt einem eigentlich erst mal so eine Ready-Made-Pipeline.
18 Das ist eine standardisierte Vorgabe, auf welcher man dann Konfigurationen vor-
19 nimmt. Dann kannst du z.B. einstellen, welche Tools du verwenden möchtest. Du
20 kannst natürlich davon ausbrechen und sagen okay, ich möchte da jetzt einen kom-
21 pletten Step überschreiben etc. Das Ziel ist jedoch den kognitiven Load so gering
22 wie möglich zu halten.

23 **Interviewer:** Wie definierst du für dich CI/CD?

24 **Experte:** Ich bin dort jetzt kein kompletter Experte, aber mein Hauptmotiv als

25 Entwickler ist, es möglichst schnelles Feedback zubekommen. Früher war es so, dass
26 ich Tests geschrieben habe und die dann einmal in der Woche ausgeführt habe. Auf-
27 grund der Verzögerung ist das natürlich nicht besonders geschickt. Bei einem guten
28 Setup mache ich eine Änderung und bekomme beim Commit direkt ein Feedback.
29 Das zweite ist natürlich, dass ich neue Produktversionen sehr schnell zum Kunden
30 bekomme. Idealerweise innerhalb von einer Woche oder vielleicht sogar manchmal
31 in einem Tag. Als ich damals in einer SAP-Partnerfirma war, haben wir manchmal
32 ein ganzes Jahr entwickelt. Dann gab es eine sehr große Testphase. Und am Ende
33 hat sich herausgestellt, dass der Kunde etwas ganz anderes haben wollte.

34 **Interviewer:** Welchen Vorteil hat es, wenn man den CI/CD-Prozess automatisiert?

35 **Experte:** Man hat die Möglichkeit, neue Features erst mal einzelnen Kunden be-
36 reitzustellen. Wenn ich dann feststelle, dass irgendwas nicht funktioniert, kann ich
37 schnell ein Rollback machen. Des Weiteren gibt es dann z.B. das Feature-Toggle. Da
38 wird eine neue Funktionalität dann hinter einer Flag versteckt. Wenn ein bestimm-
39 ter Kunde dieses Feature haben möchte, dann setzt er entsprechend die Flag.

40 **Interviewer:** Welche Art von Pipelines werden denn i.d.R. verwendet?

41 **Experte:** Ich kenne hauptsächlich die Pull-Request-Pipeline. Häufig gibt es dann
42 auch noch einmal eine Pipeline, welche einmal am Tag läuft, bei welcher dann Tests
43 gemacht werden, welche deutlich länger laufen.

44 **Interviewer:** Welchen Vorteil hat ein Artefakt-Repository?

45 **Experte:** Das Artefakt-Repository wird verwendet, um das Coding was erzeugt
46 wurde, versioniert abzulegen. Dieses wird dann in der Pipeline immer wieder verwen-
47 det, um z.B. Tests dagegen auszuführen. Mit diesen Artefakten kann man dann auch
48 noch sehr gut Rollbacks ausführen. Das heißt, wenn man merkt, dass eine neue Ver-
49 sion nicht funktioniert, kann man einfach wieder zur alten Version zurückspringen.

50 **Interviewer:** Welche Pipelines werden bei der SAP im Regelfall verwendet?

51 **Experte:** Auf der Orchestratorseite ist es so, dass ganz viel über Azure Pipelines
52 gemacht wird. Diese bietet z.B. einige Governance-Checks an, was sich gerade in der
53 Standardentwicklung als Vorteil erweist. Außerdem ist der Wartungsaufwand ein-
54 fach viel geringer. Das SAP Tools Team hat alles auf einen zentralen Blick und

55 kann entsprechend sehen, ob einer Pipeline irgendwie mehr Ressourcen zugewiesen
56 werden müssen. Zudem bekommt die SAP, weil sie Microsoft-Partner ist, auch gute
57 Konditionen bei dieser Firma. Ein weiterer Vorteil von Azure Pipelines sind Me-
58 chanismen, wie Caching oder Parallel Builds. In der internen Standardentwicklung
59 haben diese dafür gesorgt, dass die CI/CD-Prozesse um 35 Prozent beschleunigt
60 wurden. Bei Azure Pipelines werden zudem sehr viele Technologien unterstützt, was
61 insbesondere bei Unternehmen, welche einen hohen Wert auf Technologieoffenheit
62 legen, von Vorteil sein könnte. Für kleinere Teams wie das SAP Sports One emp-
63 fiehlt sich Azure Pipelines nicht. Diese sollten eher auf Jenkins setzen.

64 **Interviewer:** Welche Tests werden bei der SAP gemacht?

65 **Experte:** Bei der SAP werden durch den Produktstandard gemäß ISO 9001 ver-
66 schiedene Validierungen vorgeschrieben. Für Unit Tests gibt es dafür verschiedene
67 Frameworks. I.d.R. wird bei der SAP Q-Unit für Frontend und Mocha oder Jest
68 für das Backend verwendet. Für Q-Unit wird die Laufzeitumgebung Karma und für
69 Mocha und Jest Node benötigt. Für Frontend-Integration-Tests wird OPA5 verwen-
70 det. Dafür benötigt es ebenfalls der Laufzeitumgebung Karma. Für Integration-Tests
71 im Backend wird Newmann verwendet. Mit Newman können Postmann-Tests auto-
72 matisiert werden. Und für E2E-Tests im Frontend wird dann i.d.R. WDI5 verwendet.
73 Damit lassen sich dann ganze Anwenderszenarien testen. Das hat den Vorteil, dass
74 ich wie ein End-User teste. Nachteil ist dabei jedoch, dass ich schauen muss, dass die
75 Daten verfügbar sind, Customizings gemacht wurden etc. Um OPA5 in einer Pipe-
76 line zu automatisieren wird die Webdriver.io Laufzeitumgebung benötigt. Alle, die
77 hier genannten Laufzeitumgebungen werden durch das Project Piper ausgeliefert.
78 Aber man muss die Tests natürlich immer gezielt einsetzen. Wir haben damals in
79 unsere Pipeline E2E-Tests eingebaut und dadurch hat die Pipeline um den Faktor 3
80 länger gebraucht. Noch einmal zur zentralen Aussage der Testpyramide. Die Emp-
81 fehlung ist, möglichst viel auf den unteren Ebenen abzudecken, also mit Unit-Tests
82 und auf den oberen Ebenen nur noch das zu testen, was man nicht mit Unit- und
83 Integration-Tests validieren kann.

84 **Interviewer:** Werden denn immer alle Tests ausgeführt?

85 **Experte:** Also bei einer Pull-Request-Pipeline sollten auf jeden Fall die Unit- und
86 Integration-Tests laufen. Die System-Tests werden dann z.B. einmal am Tag aus-
87 geführt. Manche Teams verwenden auch eine parallele Ausführung von Tests und
88 führen dann eben verschiedene Szenarien gleichzeitig durch. Das läuft dann i.d.R.
89 schneller und dann können solche Tests auch beim Pull-Request ausgeführt werden.
90 Gerade die Compliance und Accessibility-Tests werden dann eher in der Delivery-
91 Pipeline durchgeführt.

92 **Interviewer:** Auf welche Integrationsaspekte muss geachtet werden?

93 **Experte:** In den bisherigen Projekten, in welchen ich gearbeitet habe, wurde die
94 Auswahl einer CI/CD-Pipeline immer sehr stark von dem Repository abhängig ge-
95 macht. Die Repositories, welche von Kunden dabei besonders oft verwendet werden,
96 sind GitHub, GitLab und BitBucket. Was auch, aber eher selten in Kundenprojekten
97 beachtet wird, ist die Integrierbarkeit in Projektmanagement-Tools. Häufig wird da-
98 bei Jira verwendet, aber i.d.R. machen die Kunden die Wahl einer CI/CD-Pipeline
99 nicht von der Unterstützung eines spezifischen Tools abhängig. Um einen Überblick
100 über die Bereitstellungsprozesse zu erhalten, müssen Experten somit nicht in den
101 Code schauen, sondern haben alles in einem zentralen Tool. SAP CI/CD unterstützt
102 das nicht, wohingegen ich gehört habe, dass es diese Möglichkeit bei Jenkins sowie
103 Azure Pipelines gibt.

104 **Interviewer:** Wie wird der Entwickler über den Erfolg der Tests informiert?

105 **Experte:** Da gibt es unterschiedliche Verfahrensweisen. Es gibt da dann z.B. ver-
106 schiedene Monitoring-Tools in der CI/CD-Pipeline. Ein anderer Weg ist, wenn man
107 die CI/CD-Pipeline über APIs in das Repository integriert. Was auch häufig ge-
108 macht wird ist, dass man die Pipelines in den SAP Alert Service integriert, sodass
109 Entwickler dann entsprechend Nachrichten über Mail oder Slack bekommen.

C.5 Kodierung der Experteninterviews

Was ist CI/CD?

Aussage	Kodierung	Experte	Zeilennummer
„Dabei habe ich einen CI-Server, der mir nach einem Push in mein zentrales Repository innerhalb kurzer Zeit ein Feedback gibt.“	CI	Experte 1	8 ff.
„Das ist die Möglichkeit, ein Feature so schnell wie möglich auf die Produktion zu überführen und für den Kunden bereitzustellen.“	CD	Experte 2	10 ff.

Verschiedene Arten von Pipelines

Aussage	Kodierung	Experte	Zeilennummer
„[Mit der CD-Pipeline] wird zentral gebaut, getestet und ggf. auch noch Sachen wie Compliance, Vulnerabilities, statische Codechecks, Integrations-Tests und Performance-Tests abgewickelt.“	Bestandteile CD-Pipeline	Experte 1	12 ff.
„Diese sollte maximal 10 bis 15 Minuten Laufzeit besitzen. So soll der Entwickler ein schnelles Feedback bekommen.“	Pull-Request-Pipeline	Experte 1	28 ff.

„ Die Pipelines werden somit deutlich verkleinert, womit schnelleres Feedback gegeben werden kann.“	Kleine Pipelines	Experte 2	29 ff.
---	------------------	-----------	--------

Deploy und Release

Aussage	Kodierung	Experte	Zeilennummer
„Das getestete Programm kann dann anschließend z.B. in ein Artefakt-Repository oder in eine Produktionsumgebung bereitgestellt werden.“	Artefakt-Repository und Produktionsumgebung	Experte 1	15 ff.
„Mit diesen Artefakten kann man dann auch noch sehr gut Rollbacks ausführen.“	Artefakt-Repository	Experte 4	47 ff.
„Kleine entwickelte Komponenten können mit Versionierung in das Artefakt-Repository bereitgestellt werden. Andere Entwickler können diese Komponente dann aus dem Artefakt-Repository herausziehen und für eigenen Entwicklungen wiederverwenden“	Komponentenwiederverwendung im Artefakt-Repository	Experte 1	40 ff.

Test

Aussage	Kodierung	Experte	Zeilennummer
„Typischerweise beginnt eine CI/CD-Pipeline mit dem Build-Stage bei welcher Unit-Tests ausgeführt werden. Für CAP werden dabei die Frameworks Mocha oder Jest verwendet.“	Unit-Tests mit SAP CAP Node	Experte 1	45 ff.
„Solche Akzeptanztests können dann z.B. Integration-Tests umfassen, welche bei CAP-Node-Anwendungen mit Newmann automatisiert werden.“	Integration-Tests mit SAP CAP Node	Experte 1	48 ff.
„I.d.R. wird bei der SAP Q-Unit für Frontend und Mocha oder Jest für das Backend verwendet.“	Unit-Tests mit SAP UI5	Experte 4	66 ff.
„Für Integration-Tests wird OPA5 verwendet.“	Integration-Tests mit SAP UI5	Experte 4	69 ff.
„UUnd für E2E-Tests im Frontend wird dann i.d.R. WDI5 verwendet.“	System-Tests mit SAP UI5	Experte 4	72 ff.

„Die Empfehlung ist, möglichst viel auf den unteren Ebenen abzudecken, also mit Unit-Tests und auf den oberen Ebenen nur noch das zu testen, was man nicht mit Unit- und Integration-Tests validieren kann.“	Zeitpunkt für Tests	Experte 4	80 ff.
--	---------------------	-----------	--------

Code-Analysen

Aussage	Kodierung	Experte	Zeilennummer
„Für statische Codeanalysen wird SonarQube und Lint verwendet.“	Statische Code-Analysen	Experte 3	45 ff.
„Für CAP Node wird in den Produktstandards das Tool Checkmarx vorgeschrieben“	SAP CAP Node	Experte 3	40 ff.
„Für SAP UI5 wird bei der statischen Code-Analyse Checkmarx verwendet.“	SAP UI5	Experte 3	41 ff.

Vorteile von kontinuierlicher Bereitstellung

Aussage	Kodierung	Experte	Zeilennummer
„So kann ich, wenn ich schnell in eine Canary-Umgebung bereitstelle, natürlich auch früher Fehler finden, was dann im Endeffekt auch deutlich günstiger wird.“	Frühe Fehlerfindung	Experte 1	23 ff.
„So ist die Gefahr, dass etwas im Produktivsystem kaputtgeht sehr gering.“	Wenig Fehler in der Produktion	Experte 2	13 ff.
„Da wird eine neue Funktionalität dann hinter einer Flag versteckt. Wenn ein bestimmter Kunde dieses Feature haben möchte, dann setzt er entsprechend die Flag.“	Feature Toggle	Experte 4	37 ff.

CI/CD-Pipeline-Tools bei der SAP

Aussage	Kodierung	Experte	Zeilennummer
„Zum einen wird der von der SAP bereitgestellte CI/CD-Service verwendet.“	SAP BTP CI/CD	Experte 1	57 ff.
„Des Weiteren gibt es Jenkins. Diese wird i.d.R. mit Project Piper verwendet.“	Jenkins	Experte 1	59 ff.

„Häufig wird für interne Projekte ebenfalls Azure Pipelines verwendet.“	Azure Pipelines	Experte 1	62 ff.
---	-----------------	-----------	--------

Aspekte für Wahl einer CI/CD-Pipeline

Aussage	Kodierung	Experte	Zeilennummer
„Für Abteilungen, welche keine DevOps-Spezialisten haben, spielt die Benutzerfreundlichkeit eine große Rolle. Da ist es zum einen wichtig, wie leicht sich die Tools warten lassen, aber auch, wie leicht sich eine Pipeline implementieren lässt.“	Intuitive Bedienbarkeit und Installation und Wartung	Experte 1	67 ff.
„Weiterhin ist wichtig zu wissen, wie flexibel man bei der Pipeline-Gestaltung sein will.“	Flexibilität	Experte 1	70 ff.
„Zudem muss natürlich auch evaluiert werden, welche Funktionalitäten, also Tests, Code-Scans und Builds auf der Pipeline ausgeführt werden sollen.“	Tests, Code-Analysen Build	Experte 1	71 ff.
„In Bezug auf die Funktionalität sollte ebenfalls evaluiert werden, auf welcher Plattform die Software bereitgestellt werden soll.“	Deploy und Release	Experte 1	73 ff.

„Insbesondere für CEA spielt ebenfalls die Skalierbarkeit eine wichtige Rolle.“	Skalierbarkeit	Experte 1	74 ff.
„Die Integration ist ebenfalls ein sehr wichtiger Aspekt bei der Auswahl einer CI/CD-Pipeline.“	Integrationsmöglichkeiten	Experte 1	88 ff.
„Dabei muss darauf geachtet werden, dass die Pipeline mit dem Repository integrierbar ist.“	Integrationsmöglichkeiten von Repositorys	Experte 1	89 ff.
„Sehr selten wird eine CI/CD-Pipeline auch in die Entwicklungsumgebung integriert.“	Integrationsmöglichkeiten von Entwicklungsumgebung	Experte 1	93 ff.
„Was auch, aber eher selten in Kundenprojekten beachtet wird, ist die Integrierbarkeit in Projektmanagement-Tools. Häufig wird dabei Jira verwendet, aber i.d.R. machen die Kunden die Wahl einer CI/CD-Pipeline nicht von der Unterstützung eines spezifischen Tools abhängig.“	Integration in Planungstools	Experte 4	96 ff.
„Für sehr großen Entwicklungsprojekte ist es ebenfalls essenziell, dass die Pipelines eine gute Performance besitzen. Somit kann Software schneller bereitgestellt werden.“	Performance	Experte 2	35 ff.

„Des Weiteren ist es essenziell, dass die CI/CD-Prozesse überwacht werden können.“	Monitoring	Experte 2	37 ff.
„Insbesondere für kleinere Kunden ist es essenziell, welche Kosten durch die Pipeline verursacht werden.“	Kosten	Experte 2	42 ff.
„Für viele Kunden ist darüber hinaus der Support wichtig. Es sollten kontinuierliche Updates, Schulungsmaterial sowie wie eine gute Dokumentation verfügbar sein.“	Administrativer Support	Experte 2	44 ff.
„Außerdem ist es für Entwickler immer vorteilhaft, wenn für die Tools eine große Community existiert.“	Administrativer Support	Experte 2	46 ff.
„Für viele Kunden spielt ebenfalls die Sicherheit eine ausschlaggebende Rolle.“	Sicherheit	Experte 1	79 ff.

SAP BTP CI/CD-Service

Aussage	Kodierung	Experte	Zeilennummer
„Was bisher noch nicht wirklich unterstützt wird, sind API-Tests.“	Keine API-Tests	Experte 1	85 ff.

„Der SAP CI/CD-Service unterstützt einen ganz normalen Git-Server. Was ebenfalls funktioniert, sind BitBucket Repositorys. “	Unterstützung von Repositorys	Experte 1	90 ff.
„Hierbei können jedoch ausschließlich Commit Events verarbeitet werden.“	Unterstützung von Commit-Events	Experte 1	92 ff.
„Aber ein direktes Monitoring der Pipeline gibt es nicht.“	Kein Monitoring für SAP CI/CD	Experte 1	104 ff.
„Eine Build-Hour kostet einen Euro.“	Kosten	Experte 1	106 ff.
„ Wir können sowohl auf Cloud Foundry als auch auf Kyma deployen.“	Deployment	Experte 1	99 ff.
„Nein leider nicht. Aber die Pipeline kann Software auf das Transport Management System bereitstellen.“	Parallel Build und SAP CTM	Experte 1	109 ff.
„Für interne Projekte darf das SAP CI/CD aufgrund der derzeitigen Produktstandards nicht verwendet werden.“	Nicht für interne Projekte	Experte 2	53 ff.
„Das SAP CI/CD-Tool lohnt sich insbesondere für Kunden, welche noch nicht viel DevOps-Expertise besitzen und auch keine teure Infrastruktur betreiben wollen.“	Für Kunden mit wenig Expertise	Experte 2	55 ff.

Azure Pipelines

Aussage	Kodierung	Experte	Zeilennummer
„Diese bietet z.B. einige Governace-Checks an, was sich gerade in der Standardentwicklung als Vorteil erweist.“	Governance-Checks	Experte 4	52 ff.
„Das SAP Tools Team hat alles auf einen zentralen Blick und kann entsprechend sehen, ob einer Pipeline irgendwie mehr Ressourcen zugewiesen werden müssen.“	Erhöhte Flexibilität	Experte 4	54 ff.
„Zudem bekommt die SAP, weil sie Microsoft-Partner ist, auch gute Konditionen bei dieser Firma.“	Kosten	Experte 4	56 ff.
„Ein weiterer Vorteil von Azure Pipelines sind Mechanismen, wie Caching oder Parallel Builds.“	Parallel Builds und Caching	Experte 4	57 ff.
„Bei Azure Pipelines werden zudem sehr viele Technologien unterstützt, was insbesondere bei Unternehmen, welche einen hohen Wert auf Technologieoffenheit legen, von Vorteil sein könnte.“	Technologieoffenheit	Experte 4	60 ff.

Security

Aussage	Kodierung	Experte	Zeilennummer
„Früher gab es dabei immer einen Security-Experten, welcher sich vor der Auslieferung um alles kümmern musste.“	Security damals	Experte 3	7 ff.
„Security sollte nicht mehr nur von einem Spezialisten behandelt werden. Vielmehr sollte dies als Kollektiv vorangetrieben werden. Jeder muss bei der Entwicklung seiner Funktionalitäten schon so früh wie möglich schauen, ob alle sicherheitsrelevanten Aspekte eingehalten wurden.“	Security heute	Experte 3	11 ff.
„Das wird dann i.d.R. durch Automatisierung gemacht. Es wird dabei schon sehr lange mit Security-Tools gearbeitet.“	Automatisierung mit Tools	Experte 3	32 ff.
„Dort wird insbesondere OS-Scanning betrieben.“	Statische Code-Analysen	Experte 3	27 ff.

<p>„Dort werden dann auch tatsächlich UI-Elemente, APIs sowie Datenbanken gescannt.</p> <p>Damit können im Produktionssystem leichter Scripting-Attacken oder SQL-Injections verhindert werden.“</p>	<p>Dynamic Application Security Testing</p>	<p>Experte 3</p>	<p>29 ff.</p>
<p>„Für CAP Node wird in den Produktstandards das Tool Checkmarx vorgeschrieben. Für Open-Source ist das Tool Whitesource vorgeschrieben.“</p>	<p>Security-Scans für SAP CAP Node</p>	<p>Experte 3</p>	<p>40 ff.</p>
<p>„Für SAP UI5 wird bei der statischen Code-Analyse Checkmarx verwendet.“</p>	<p>SAP UI5</p>	<p>Experte 3</p>	<p>41 ff.</p>

C.6 Expertengewichtung 1

Interviewpartner: Softwarearchitekt SAP DTS Integration (Experte 5)

Datum: 27.03.2023

Interview-Medium: Microsoft-Teams

Funktionalität	Integrationsmöglichkeiten	Kosten	Skalierbarkeit	Performance	Flexibilität	Support	Sicherheit	Benutzerfreundlichkeit	Lokale Gewichtung
Funktionalität	1		2	2	1	2	2	2	0,1852
Integrationsmöglichkeiten	1	1		2	1	2	1	2	0,1605
Kosten	0	1	1	2	2	2	1	2	0,1481
Skalierbarkeit	0	0	1	2	1	2	1	2	0,1235
Performance	0	0	0	1	1	0	0	1	0,0370
Flexibilität	1	1	0	1	1	1	1	2	0,1111
Support	0	0	0	0	2	1	1	2	0,0864
Sicherheit	0	1	1	2	1	1	1	2	0,1235
Benutzerfreundlichkeit	0	0	0	1	0	0	0	1	0,0247
									1,000

Funktionalität	Tests	Build	Deploy/Release	Monitoring	Code-Analysen	Lokale Gewichtung
Tests	1	1	1	2	1	0,2400
Build	1	1	2	2	2	0,3200
Deploy/Release	1	0	1	2	2	0,2400
Monitoring	0	0	0	1	0	0,0400
Code-Analysen	1	0	0	2	1	0,1600
						1,0000

Integrationsmöglichkeiten	Entwicklungsumgebung	Repository	Planungssoftware	Lokale Gewichtung
Entwicklungsumgebung	1	0	2	0,3333
Repository	2	1	2	0,5556
Planungssoftware	0	0	1	0,1111
				1,000

Benutzerfreundlichkeit				
Installation und Wartung	1	1	1	0,5000
Intuitive Bedienbarkeit	1	1	1	0,5000
				1,0000

Performance				
Integration-Zeit	1	1	1	0,5000
Delivery-Zeit	1	1	1	0,5000
				1,000

Support				
Administrativer Support	1	2	1	0,7500
Community-Support	0	1	1	0,2500
				1,000

Endfaktoren	
Kriterien	Globale Gewichtung
Test	0,0444
Code-Analysen	0,0296
Build	0,0593
Deploy und Release	0,0444
Monitoring	0,0074
Integration in Repository	0,0892
Integration in Entwicklungsumgebung	0,0535
Integration in Planungssoftware	0,0178
Kosten	0,1481
Skalierbarkeit	0,1235
Integration-Zeit	0,0185
Delivery-Zeit	0,0185
Flexibilität	0,1111
Administrativer Support	0,0648
Community-Support	0,0216
Sicherheit	0,1235
Installation und Wartung	0,0123
Intuitive Bedienbarkeit	0,0123
	1,0000

1 **Interviewer:** Für dich besonders wichtig ist die Funktionalität. Kannst du das
2 begründen?

3 **Experte:** Wenn ich eine Pipeline einbaue, dann ist für mich besonders wichtig, dass
4 die Pipelines bestimmte Funktionalitäten, wie z.B. Tests abdecken. Weniger gewich-
5 tig ist da z.B. die Benutzerfreundlichkeit. Ich bin der Meinung, dass ich eine Pipeline
6 einmalig einrichte. Da spielt es dann auch weniger die Rolle, wie viel Aufwand das
7 beim initialen Einrichten erfordert hat. Beim Implementieren der Pipelines sieht
8 das meiner Meinung nach ein wenig anders aus. So sollte ein Unternehmen zunächst
9 damit beginnen; einfache Prozesse in der CI/CD-Pipeline zu automatisieren. Dann
10 kann schrittweise immer mehr in die Pipeline eingebunden werden.

11 **Interviewer:** Von den Subkriterien der Funktionalität war für dich die Build-
12 Funktionalität besonders essenziell. Kannst du das begründen?

13 **Experte:** Bei Cloud-Native-Projekten werden i.d.R. verschiedene Sprachen und ver-
14 schiedene Frameworks verwendet. Die Pipeline sollte da einfach alle Build-Packages
15 unterstützen. So muss ich dann nicht hergehen und manuell irgendwelche Deploy-
16 ments ausführen. Tests waren für mich auch sehr wichtig. Es ist wichtig, dass die
17 Tests ausgeführt werden, bevor Code in dem Main-Branch zusammengeführt wird.
18 Natürlich kann ich auch Tests abseits von einer CI/CD-Pipeline ausführen. Bei ei-
19 nem größeren Team von Entwickler ist das jedoch kaum noch kontrollierbar.

20 **Interviewer:** Kommen wir zu den Integrationsmöglichkeiten. Deiner Meinung nach
21 ist die Integration eines Repositorys sehr wichtig. Woran liegt das?

22 **Experte:** Oft ist es so, dass sich der Kunde auf eine oder zwei CI/CD-Tools und
23 ein oder zwei Repositorys einschießt. Gerade, wenn, da die Abhängigkeit mit dem
24 Repository besteht, sollte das natürlich in die CI/CD-Pipeline integrierbar sein.

25 **Interviewer:** Kannst du begründen, warum für dich die Integration- und Delivery-
26 Zeit gleich wichtig sind.

27 **Experte:** Aus meiner Sicht gibt es da kein wichtigeres Kriterium, da sowohl CI als
28 auch CD im Hintergrund läuft. Somit ist es für mich jetzt nicht so wichtig, falls eine
29 der beiden Pipelines länger durchläuft.

30 **Interviewer:** Warum ist für dich der administrative Support wichtiger als der

31 Community-Support?

32 **Experte:** Für mich ist es eben sehr wichtig, dass es eine gute Dokumentation
33 gibt. So kann ich z.B., bevor ich eine Pipeline installiere, schon abschätzen, wie
34 gut bestimmte Aspekte funktionieren und ich bin nicht davon abhängig, dass ich
35 durch Community-Foren irgendwelche Workarounds bekomme. Außerdem ist es mir
36 natürlich auch sehr wichtig, dass die Lösung immer auf dem neusten Stand der Tech-
37 nik ist.

38 **Interviewer:** Kannst du noch einmal deine Entscheidung zur Sicherheit begründen?

39 **Experte:** Also es gehört natürlich einfach zur Developer-Experience dazu, wenn
40 man bestimmte Authentifizierungs- und Autorisierungskonzepte hat. Dann ist es
41 natürlich aber auch schon wichtig, dass die Pipeline an sich sicher ist. Gerade die
42 Pipeline bietet natürlich eine sehr gute Möglichkeit, feindliche Programme in eine
43 Produktionsumgebung einzuführen.

44

C.7 Expertengewichtung 2

Interviewpartner: Full-Stack-Entwickler SAP DTS Integration (Experte 6)

Datum: 21.03.2023

Interview-Medium: Microsoft-Teams

Funktionalität	Integrationsmöglichkeiten	Kosten	Skalierbarkeit	Performance	Flexibilität	Support	Sicherheit	Benutzerfreundlichkeit	Lokale Gewichtung
Funktionalität	1	1	2	1	1	1	1	1	0,1235
Integrationsmöglichkeiten	1	1	2	1	1	0	1	1	0,1235
Kosten	0	0	2	1	0	0	0	0	0,0123
Skalierbarkeit	1	0	1	0	0	0	0	0	0,0494
Performance	1	1	2	1	0	0	0	0	0,1111
Flexibilität	1	1	2	1	1	0	1	0	0,1235
Support	1	2	2	2	2	1	1	1	0,1728
Sicherheit	1	2	2	2	1	1	1	1	0,1358
Benutzerfreundlichkeit	1	1	2	2	1	1	1	1	0,1481
									1,000

Funktionalität	Tests	Build	Deploy/Release	Monitoring	Code-Analysen	Lokale Gewichtung
Tests	1	2	2	2	2	0,3600
Build	0	1	0	0	0	0,0400
Deploy/Release	0	2	1	2	0	0,2000
Monitoring	0	2	0	1	2	0,2000
Code-Analysen	0	2	2	0	1	0,2000
						1,0000

Integrationsmöglichkeiten	Entwicklungsumgebung	Repository	Planungssoftware	Lokale Gewichtung
Entwicklungsumgebung	1	0	2	0,3333
Repository	2	1	2	0,5556
Planungssoftware	0	0	1	0,1111
				1,000

Endfaktoren	
Kriterien	Globale Gewichtung
Test	0,0444
Code-Analysen	0,0247
Build	0,0049
Deploy und Release	0,0247
Monitoring	0,0247
Integration in Repository	0,0686
Integration in Entwicklungsumgebung	0,0412
Integration in Planungssoftware	0,0137
Kosten	0,0123
Skalierbarkeit	0,0494
Integration-Zeit	0,0833
Delivery-Zeit	0,0278
Flexibilität	0,1235
Administrativer Support	0,0432
Community-Support	0,1296
Sicherheit	0,1358
Installation und Wartung	0,0741
Intuitive Bedienbarkeit	0,0741
	1,0000

1 **Interviewer:** In Tabelle 1 ist für dich besonders wichtig der Support und weni-
2 ger wichtig sind für dich die Kosten. Warum?

3 **Experte:** Ich als Entwickler habe die Erfahrung, dass ein Tool noch so toll sein
4 kann. Wenn keine gute Dokumentation vorhanden ist, dann hilft mir das als Ent-
5 wickler nicht viel. Die Kosten sind mir aus Entwicklersicht egal. Da sind mir erst
6 mal die anderen Kriterien wichtiger.

7 **Interviewer:** In Tabelle 2 waren dir die Tests besonders wichtig. Kannst du das
8 begründen.

9 **Experte:** Es spart mir sehr viel Zeit, wenn Tests automatisiert werden. Für mich
10 gehört das automatisierte Ausführen von Tests auch zu den Hauptzwecken einer
11 CI/CD-Pipeline. So kann ich frühzeitig erkennen, wenn eine Änderung etwas ka-
12 putt macht.

13 **Interviewer:** Kommen wir zu den Integrationsmöglichkeiten. Warum ist dir die In-
14 tegration in Repositorys besonders wichtig?

15 **Experte:** Ich sehe, dass als eine sehr grundlegende Funktion. Wenn meine CI/CD-
16 Pipeline nicht mit dem Repository integrierbar ist, dann wird das gesamte Konzept
17 einer Pipeline hinfällig. Die Planungssoftware war mir hingegen nicht so wichtig, da
18 ich als Entwickler mit so etwas kaum arbeite.

19 **Interviewer:** Kannst du deine Entscheidungen zur Performance begründen?

20 **Experte:** Für mich ist die Integration-Zeit deutlich wichtiger, einfach um eine besse-
21 re Developer-Experience zu bekommen. Wenn ich ein Pull-Request aufmache, dann
22 möchte ich auch schnell Feedback bekommen. So kann ich dann evaluieren, ob alles
23 passt oder eben nicht.

24 **Interviewer:** Kannst du deine Entscheidungen zum Support begründen?

25 **Experte:** Ich finde den Community-Support am wichtigsten. Dies ist als Entwick-
26 ler die beste Möglichkeit, Zugriff auf neues Wissen zu erlangen. Weniger geeignet
27 wäre, wenn ich jedes Mal mit jemandem telefonieren müsste oder immer ein neues
28 Ticket aufmachen müsste. Weiterhin ist es sehr gut, wenn von einer Community
29 Plug-ins bereitgestellt werden. Damit kann ich den Funktionsumfang erweitern. Je-
30 doch besteht hier immer die Gefahr, dass Plug-ins Sicherheitslücken besitzen oder

31 irgendwann nicht mehr richtig gewartet werden.

32 **Interviewer:** Wie sieht es mit dem Punkt der Sicherheit aus?

33 **Experte:** Ich finde das Authentifizierungs- und Autorisierungskonzept sehr wich-
34 tig, da dies mit der Developer-Experience zusammenhängt. Damit komme ich eben
35 in meinem Entwickleralltag am meisten in Berührung. Wenn die Plattform bspw.
36 SSO-enabled ist, dann ist das für mich als Programmierer sehr gemütlich.

37 **Interviewer:** Kannst du deine Entscheidung zur Benutzerfreundlichkeit begründen?

38 **Experte:** Sowohl mit Wartung bzw. Installation und mit der intuitiven Bedienbar-
39 keit kommt man gelegentlich in Berührung. Das sollte deshalb beides einigermaßen
40 gut funktionieren.

41

C.8 Expertengewichtung 3

Interviewpartner: Frontend-Test-Entwickler SAP Hyperspace (Experte 4)

Datum: 22.03.2023

Interview-Medium: Microsoft-Teams

	Funktionalität	Integrationsmöglichkeiten	Kosten	Skalierbarkeit	Performance	Flexibilität	Support	Sicherheit	Benutzerfreundlichkeit	Lokale Gewichtung
Funktionalität	1	2	2	2	2	2	2	0	2	0,1852
Integrationsmöglichkeiten	0	1	1	1	2	1	2	0	2	0,1358
Kosten	0	0	1	0	2	2	1	0	1	0,0741
Skalierbarkeit	0	1	2	1	1	1	2	0	2	0,1358
Performance	0	0	0	1	1	1	2	0	2	0,0988
Flexibilität	0	1	1	0	0	1	1	0	1	0,0617
Support	0	0	0	0	0	0	1	1	0	0,0370
Sicherheit	2	2	2	2	2	2	2	1	2	0,2099
Benutzerfreundlichkeit	0	0	1	0	0	1	2	0	1	0,0617
										1,000

Funktionalität	Tests	Build	Deploy/Release	Monitoring	Code-Analysen	Lokale Gewichtung
Tests	1	1	2	2	1	0,2800
Build	1	1	2	2	1	0,2800
Deploy/Release	0	0	1	1	1	0,1200
Monitoring	0	0	1	1	1	0,1200
Code-Analysen	1	1	1	1	1	0,2000
						1,0000

Integrationsmöglichkeiten	Entwicklungsumgebung	Repository	Planungssoftware	Lokale Gewichtung
Entwicklungsumgebung	1	0	2	0,3333
Repository	2	1	2	0,5556
Planungssoftware	0	0	1	0,1111
				1,000

Performance				
	Integration-Zeit		Delivery-Zeit	Lokale Gewichtung
	1	2		0,7500
	0	1		0,2500
				1,000

Support				
	Administrativer Support	Community-Support		Lokale Gewichtung
	1	0		0,2500
	2	1		0,7500
				1,000

Benutzerfreundlichkeit				
	Installation und Wartung	Intuitive Bedienbarkeit		Lokale Gewichtung
	1	0		0,2500
	2	1		0,7500
				1,0000

Endfaktoren	
Kriterien	Globale Gewichtung
Test	0,0519
Code-Analysen	0,0370
Build	0,0519
Deploy und Release	0,0222
Monitoring	0,0222
Integration in Repository	0,0754
Integration in Entwicklungsumgebung	0,0453
Integration in Planungssoftware	0,0151
Kosten	0,0741
Skalierbarkeit	0,1358
Integration-Zeit	0,0741
Delivery-Zeit	0,0247
Flexibilität	0,0617
Administrativer Support	0,0093
Community-Support	0,0278
Sicherheit	0,2099
Installation und Wartung	0,0154
Intuitive Bedienbarkeit	0,0463
	1,0000

1 **Interviewer:** Besonders wichtig ist für dich die Sicherheit. Kannst du das be-
2 gründen?

3 **Experte:** Sicherheitslücken in unseren Systemen würden einen sehr großen Image-
4 schaden verursachen. Deswegen ist das für mich das absolute Top-Thema. Gerade
5 wenn wir neue Software bereitstellen, darf es nicht passieren, dass eventuell feindli-
6 che Programme eingeschleust werden. Nicht so wichtig ist für mich der Support, da
7 ein Tool lieber eine hohe Benutzerfreundlichkeit besitzen sollte und wobei dann gar
8 kein Support nötig wäre.

9 **Interviewer:** Warum ist für dich der administrative Support wichtiger als der
10 Community-Support?

11 **Experte:** Es ist viel wichtiger, dass man eine Community besitzt, welche Fragen
12 beantworten kann. Dies ist eigentlich die einzige Möglichkeit Support skalierbar zu
13 machen. Man wird niemals ein Support-Team besitzen, was groß genug ist, alle
14 Fragen zu beantworten. Man könnte den administrativen Support also nicht gut
15 skalieren.

16 **Interviewer:** Im Kriterium der Funktionalität ist für dich die Test- und Build-
17 Funktionalität besonders wichtig. Warum?

18 **Experte:** Test ist natürlich mein Hintergrund, da ich mich während meiner opera-
19 tiven Arbeit sehr viel damit beschäftige. Es ist für mich einfach wichtig, dass ich
20 etwas ausliefere, was auch validiert ist.

21 **Interviewer:** Warum ist für dich die Integration-Zeit wichtiger als die Delivery-
22 Zeit?

23 **Experte:** Es kann eben aus Entwicklersicht nicht sein, dass ich eine Änderung im
24 Code mache und dann erst einmal eine halbe Stunde warten muss, bis ich Feedback
25 bekomme. Da geht die Motivation im Team verloren. Und es stapeln sich einfach
26 die Änderungen, bevor ich den Code dann tatsächlich ausliefere.

27 **Interviewer:** Kannst du noch einmal deine Entscheidung zur Sicherheit begründen?

28 **Experte:** Authentifizierung ist natürlich wichtig, damit jemand unberechtigtes ei-
29 genständig ein Deployment durchführen kann.

30 **Interviewer:** Für dich war die Installation und Wartung weniger wichtig als die

31 intuitive Bedienbarkeit?

32 **Experte:** Installation und Wartung betrifft mich einmal beim Setup, während die
33 intuitive Bedienbarkeit kontinuierlich wichtig anfällt, da mit dieser ja z.B. auch das
34 Implementieren einer Pipeline gemeint ist.

35

C.9 Expertengewichtung 4

Interviewpartner: Backend-Test-Entwickler SAP DTS Integration (Experte 7)

Datum: 22.03.2023

Interview-Medium: Microsoft-Teams

	Funktionalität	Integrationsmöglichkeiten	Kosten	Skalierbarkeit	Performance	Flexibilität	Support	Sicherheit	Benutzerfreundlichkeit	Lokale Gewichtung
Funktionalität	1	1	2	1	1	1	1	1	1	0.1235
Integrationsmöglichkeiten	1	1	1	2	1	1	0	1	1	0.1235
Kosten	0	0	1	0	0	0	0	0	0	0.0123
Skalierbarkeit	1	0	2	1	0	0	0	0	0	0.0494
Performance	1	1	2	2	1	1	0	1	0	0.1111
Flexibilität	1	1	2	2	1	1	0	1	1	0.1235
Support	1	2	2	2	2	2	1	1	1	0.1728
Sicherheit	1	1	2	2	1	1	1	1	1	0.1358
Benutzerfreundlichkeit	1	1	2	2	2	1	1	1	1	0.1481
										1,000

Funktionalität	Tests	Build	Deploy/Release	Monitoring	Code-Analysen	Lokale Gewichtung
Tests	1	2	2	2	2	0.3600
Build	0	1	0	0	2	0.2000
Deploy/Release	0	2	1	2	2	0.2800
Monitoring	0	0	0	1	2	0.1200
Code-Analysen	0	0	0	0	1	0.0400
						1,0000

Integrationsmöglichkeiten	Entwicklungsumgebung	Repository	Planungssoftware	Lokale Gewichtung
Entwicklungsumgebung	1	0	1	0.2222
Repository	2	1	1	0.4444
Planungssoftware	1	1	1	0.3333
				1,000

Performance					
		Integration-Zeit		Delivery-Zeit	Lokale Gewichtung
Integration-Time		1	2		0,7500
Delivery-Time		0	1		0,2500
					1,000

Support					
		Administrativer Support		Community-Support	Lokale Gewichtung
Administrativer Support		1	0		0,2500
Community-Support		2	1		0,7500
					1,000

Benutzerfreundlichkeit					
		Installation und Wartung		Intuitive Bedienbarkeit	Lokale Gewichtung
Installation und Wartung		1	0		0,2500
Intuitive Bedienbarkeit		2	1		0,7500
					1,0000

Endfaktoren	
Kriterien	Globale Gewichtung
Test	0,0711
Code-Analysen	0,0079
Build	0,0395
Deploy und Release	0,0553
Monitoring	0,0237
Integration in Repository	0,0823
Integration in Entwicklungsumgebung	0,0412
Integration in Planungssoftware	0,0617
Kosten	0,1235
Skalierbarkeit	0,1358
Integration-Zeit	0,0556
Delivery-Zeit	0,0185
Flexibilität	0,1605
Administrativer Support	0,0123
Community-Support	0,0370
Sicherheit	0,0617
Installation und Wartung	0,0031
Intuitive Bedienbarkeit	0,0093
	1,0000

1 **Interviewer:** Fangen wir mit dem Subkriterium Funktionalität an. Kannst du
2 deine Entscheidungen bitte begründen.

3 **Experte:** Also für mich sind fast alle Funktionalitäten gleichgewichtig. Ich benötige
4 alle diese Funktionalitäten, weil meine CI/CD-Pipeline sonst nicht sonderlich nützlich
5 ist. Zu den Tests, der Hauptgrund der CI/CD-Pipeline ist es eigentlich Tests zu au-
6 tomatisieren. Ohne Build, Deploy und Release funktioniert meine Pipeline nicht.
7 Ohne Monitoring kann ich nicht evaluieren, ob etwas fehlschlägt oder nicht. Code-
8 Analysen sind hingegen bei Kundenprojekten oft nicht verpflichtend.

9 **Interviewer:** Machen wir mit den Integrationsmöglichkeiten weiter. Für dich waren
10 sowohl die Integration in das Repository als auch in eine Planungssoftware wichtig.
11 Warum?

12 **Experte:** Ich benötige auf jeden Fall mein Code für die Pipeline. Deswegen ist die
13 Integration in das Repository eine essenzielle Funktionalität. Planungssoftware ist
14 auch sehr wichtig, da das Business nicht in den Code reinschaut, sondern in eine
15 Planungssoftware. In vielen Projekten, in den ich war, werden, die Ergebnisse der
16 Code-Analysen unmittelbar in der Planungssoftware angezeigt.

17 **Interviewer:** Für dich war die Integration-Zeit wichtiger als die Delivery-Zeit.
18 Warum?

19 **Experte:** Die Integration-Zeit ist sehr wichtig, um einen schnellen Feedback-Zyklus
20 zu haben. Außerdem habe ich bereits in vielen Projekten gearbeitet, die ein Blue-
21 Green-Deployment verwendet haben. So konnte nach Validierung lediglich auf eine
22 neue Version umgeschaltet werden, wobei der Entwickler nicht durchgehend den
23 Delivery-Prozess beobachten muss.

24 **Interviewer:** Warum ist für dich sowohl der Community-Support als auch der ad-
25 ministrative Support gleichgewichtig?

26 **Experte:** Natürlich ist Community-Support sehr wichtig. Andererseits es natürlich
27 auch sehr wichtig, dass eine gute Dokumentation und passende Schulungsunterlagen
28 bereitstehen.

29 **Interviewer:** Kannst du deine Entscheidung zur Benutzerfreundlichkeit begründen?

30 **Experte:** Eigentlich ist es so, dass man ein Pipeline-System einmal aufsetzt und

31 dieses dann nicht mehr sonderlich viel Konfiguration benötigt. Was i.d.R. häufiger
32 gemacht wird, insbesondere in einer Composable-Enterprise-Architektur ist das Auf-
33 setzen von Pipelines. Deshalb ist die intuitive Bedienbarkeit einfach sehr wichtig.

34 **Interviewer:** Noch einmal zu den Kriterien auf oberster Ebene. Warum sind dir
35 Sicherheit und Funktionalität so wichtig?

36 **Experte:** Das Bereitstellen von Software schöpft Wert für das Unternehmen. Des-
37 wegen ist es einfach wichtig, dass keiner in meine Pipelines eingreifen kann und die-
38 sen Prozess stören kann. Funktionalität ist natürlich essenziell, dass mein Pipeline-
39 System auch das abdecken kann, was dann letztendlich benötigt wird.

40

C.10 Expertengewichtung 5

Interviewpartner: Product Management CLM (Experte 8)

Datum: 22.03.2023

Interview-Medium: Microsoft-Teams

	Funktionalität	Integrationsmöglichkeiten	Kosten	Skalierbarkeit	Performance	Flexibilität	Support	Sicherheit	Benutzerfreundlichkeit	Lokale Gewichtung
Funktionalität	1	2	2	2	2	2	1	0	2	0,1728
Integrationsmöglichkeiten	0	1	1	2	2	2	2	2	2	0,1852
Kosten	0	0	1	0	2	1	2	2	2	0,1235
Skalierbarkeit	0	0	2	1	2	0	2	2	2	0,1358
Performance	0	0	0	0	1	0	2	1	2	0,0741
Flexibilität	0	0	1	2	2	1	2	1	2	0,1358
Support	1	0	0	0	0	0	1	1	2	0,0617
Sicherheit	2	0	0	0	1	1	0	1	2	0,0988
Benutzerfreundlichkeit	0	0	0	0	0	0	0	0	1	0,0123
										1,000

Funktionalität	Tests	Build	Deploy/Release	Monitoring	Code-Analysen	Lokale Gewichtung
Tests	1	2	2	2	2	0,3600
Build	0	1	0	2	2	0,2000
Deploy/Release	0	2	1	2	2	0,2800
Monitoring	0	0	0	1	2	0,1200
Code-Analysen	0	0	0	0	1	0,0400
						1,0000

Integrationsmöglichkeiten	Entwicklungsumgebung	Repository	Planungssoftware	Lokale Gewichtung
Entwicklungsumgebung	1	0	1	0,2222
Repository	2	1	1	0,4444
Planungssoftware	1	1	1	0,3333
				1,000

Performance					
	Integration-Zeit	1	Delivery-Zeit	2	Lokale Gewichtung
					0,7500
	Delivery-Zeit	0		1	0,2500
					1,000

Support					
	Administrativer Support	1	Community-Support	0	Lokale Gewichtung
					0,2500
	Community-Support	2		1	0,7500
					1,000

Benutzerfreundlichkeit					
	Installation und Wartung	1	Intuitive Bedienbarkeit	1	Lokale Gewichtung
					0,5000
	Intuitive Bedienbarkeit	1		1	0,5000
					1,0000

Endfaktoren	
Kriterien	Globale Gewichtung
Test	0,0622
Code-Analysen	0,0069
Build	0,0346
Deploy und Release	0,0484
Monitoring	0,0207
Integration in Repository	0,0823
Integration in Entwicklungsumgebung	0,0412
Integration in Planungssoftware	0,0617
Kosten	0,1235
Skalierbarkeit	0,1358
Integration-Zeit	0,0556
Delivery-Zeit	0,0185
Flexibilität	0,1358
Administrativer Support	0,0154
Community-Support	0,0463
Sicherheit	0,0988
Installation und Wartung	0,0062
Intuitive Bedienbarkeit	0,0062
	1,0000

1 **Interviewer:** Fangen wir auf oberster Ebene an. Besonders wichtig war für dich
2 die Integration. Warum?

3 **Experte:** Kunden wollen insbesondere, dass die CI/CD-Tools in ihre Prozesse inte-
4 grierbar sind. Da spielt es natürlich eine sehr große Rolle, welches Repository diese
5 verwenden. Sicherheit kann dann natürlich auch ein K.O.-Kriterium sein. Wenn
6 keine angemessenen Sicherheitsrichtlinien vorhanden sind, kann das natürlich ein
7 Grund sein eine Pipeline nicht zu wählen.

8 **Interviewer:** Machen wir mit dem Subkriterium Funktionalität weiter.

9 **Experte:** Der Build war für mich sehr wichtig. Das liegt einfach daran, dass man
10 ohne den Build gar nicht erst weiter kommt. Natürlich ist der Hauptgrund von
11 CI/CD-Pipelines automatisierte Tests durchzuführen, jedoch funktioniert das ohne
12 den Build erst gar nicht. Es gibt einige Kunden, die auch ganz stark auf Com-
13 pliance achten, weswegen Code-Analysen schon auch gleichwertig wie die Test-
14 Funktionalität ist. Was Deploy und Release angeht, es gibt auch einige Kunden,
15 die auch darauf verzichten.

16 **Interviewer:** Kommen wir zur Integration. Für dich ist die Integration in das Re-
17 pository sehr wichtig. Warum?

18 **Experte:** Das gehört meiner Meinung nach zur Voraussetzung. Was die Planungs-
19 software angeht, haben die Kunden meistens isolierte Lösungen.

20 **Interviewer:** In der Kategorie der Performance war dir die Integration-Zeit wich-
21 tiger. Warum?

22 **Experte:** Meiner Erfahrung nach war es für die Kunden oft in Ordnung, wenn die
23 Delivery-Zeit ein wenig länger dauert. Das liegt auch daran, dass vor dem Deploy
24 noch oft manuelle Schritte gemacht werden.

25 **Interviewer:** Kannst du deine Entscheidung zum Support begründen?

26 **Experte:** Ich habe den administrativen Support höher gewertet, da ich die Erfah-
27 rung gemacht habe, dass Kunden sehr viel Wert auf die SLAs legen. So weiß der
28 Kunde natürlich genau, dass er sich auch am Wochenende melden kann, wenn er ein
29 Problem hat und dann auch entsprechend eine Antwort bekommt.

30 **Interviewer:** Kannst du auch noch deine Entscheidung zur Benutzerfreundlichkeit

31 begründen.

32 **Experte:** Ich habe die Intuitive Bedienbarkeit sowie Installation und Wartung auf
33 eine Wichtigkeitsstufe gesetzt. Zum einen ist es natürlich so, dass die intuitive Be-
34 dienbarkeit etwas ist, mit welchem man alltäglich zu tun hat. Aber gerade bezüglich
35 des Wartens von komplexen Infrastrukturen, war es für viele Kunden der Grund sich
36 dann letztendlich für eine SaaS-Lösung zu entscheiden.

37

C.11 Expertengewichtung Durchschnitt

	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt
Funktionalität	0,1852	0,1235	0,1852	0,1975	0,1728	0,1728
Integrationsmöglichkeiten	0,1605	0,1235	0,1358	0,1852	0,1852	0,1580
Kosten	0,1481	0,0123	0,0741	0,1235	0,1235	0,0963
Skalierbarkeit	0,1235	0,0494	0,1358	0,1358	0,1358	0,1160
Performance	0,0370	0,1111	0,0988	0,0741	0,0741	0,0790
Flexibilität	0,1111	0,1235	0,0617	0,1605	0,1358	0,1185
Support	0,0864	0,1728	0,0370	0,0494	0,0617	0,0815
Sicherheit	0,1235	0,1358	0,2099	0,0617	0,0988	0,1259
Benutzerfreundlichkeit	0,0247	0,1481	0,0617	0,0123	0,0123	0,0519
						1

	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt
Funktionalität						
Tests	0,2400	0,3600	0,2800	0,3600	0,3600	0,3200
Build	0,3200	0,0400	0,2800	0,2000	0,2000	0,2080
Deploy/Release	0,2400	0,2000	0,1200	0,2800	0,2800	0,2240
Monitoring	0,0400	0,2000	0,1200	0,1200	0,1200	0,1200
Code-Analysen	0,1600	0,2000	0,2000	0,0400	0,0400	0,1280
						1

	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt
Integrationsmöglichkeiten						
Entwicklungsumgebung	0,3333	0,3333	0,3333	0,2222	0,2222	0,2889
Repository	0,5556	0,5556	0,5556	0,4444	0,4444	0,5111
Planungssoftware	0,1111	0,1111	0,1111	0,3333	0,3333	0,2000
						1

Performance									
	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt			
Integration-Zeit	0,5000	0,7500	0,7500	0,7500	0,7500	0,7000			
Delivery-Zeit	0,5000	0,2500	0,2500	0,2500	0,2500	0,3000			
						1			

Support									
	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt			
Administrativer Support	0,7500	0,2500	0,2500	0,2500	0,2500	0,3500			
Community-Support	0,2500	0,7500	0,7500	0,7500	0,7500	0,6500			
						1			

Benutzerfreundlichkeit									
	Expertengewichtung 1	Expertengewichtung 2	Expertengewichtung 3	Expertengewichtung 4	Expertengewichtung 5	Durchschnitt			
Installation und Wartung	0,5000	0,5000	0,2500	0,2500	0,5000	0,4000			
Intuitive Bedienbarkeit	0,5000	0,5000	0,7500	0,7500	0,5000	0,6000			
						1			

