



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

Gender Disclaimer

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

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 Composable-Enterprises (CEs) 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

Abkürzungsverzeichnis	VII
Abbildungsverzeichnis	IX
Tabellenverzeichnis	X
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-Tools unter An- wendung 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	38

5	Entwicklung einer Handlungsempfehlung	54
6	Schlussbetrachtung	60
6.1	Fazit und kritische Reflexion	60
6.2	Ausblick auf zukünftige Entwicklungen	62
	Literaturverzeichnis	XI
	Anhang	XXIII

Abkürzungsverzeichnis

AHP	Analytischer Hierarchieprozess
CD	Continuous Delivery
CE	Composable-Enterprise
CEA	Composable-Enterprise-Architektur
CI	Continuous Integration
CI/CD	Continuous Integration and Continuous Delivery
CIO	Chief Information Officer
CRM	Customer-Relationship-Management
DAST	Dynamic Application Security Testing
DevOps	Development & Operations
DoD	Definition of Done
E2E-Test	End-to-End-Test
EDA	Event-driven Architecture
ERP	Enterprise-Ressourcen-Planning
IaaS	Infrastructure-as-a-Service
KI	Künstliche Intelligenz
MACH	Microservice, API, Cloud-native, Headless
MTA	Multi-Target Application
NIST	National Institute of Standards and Technology
PaaS	Platform-as-a-Service
PBC	Packaged-Business-Capability
SaaS	Software-as-a-Service

SAP BAS	SAP Business Application Studio
SAP BTP	SAP Business Technology Platform
SAP CI/CD	SAP Continuous Integration and Delivery
SAP CTM	SAP Cloud Transport Management
SAP DTS	SAP Data Technology Services
SAST	Static Application Security Testing
SSO	Single-Sign-On
TCO	Total Cost of Ownership
TTM	Time-To-Market
XP	Extreme Programming

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	13
7	Integration der CI/CD-Pipeline mit dem Versionskontrollsystem . . .	13
8	Hierarchische Darstellung von Softwaretests	15
9	Strategien zur Bereitstellung von Software	19
10	Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP	23
11	Gewichtungsfaktoren der AHP-Entscheidungskriterien	36
12	SAP Cloud Transport Management	40
13	Pipeline-Monitoring mit Kibana und Jenkins	42
14	CEA-Szenario für Performance-Tests	45
15	Modularer Aufbau einer CI/CD-Pipeline	48
16	Entscheidungsbaum für die Wahl eines CI/CD-Pipeline-Tools	58

Tabellenverzeichnis

1	Exemplarische Darstellung der Paarvergleichsmatrix im AHP	24
2	AHP-Entscheidungsstruktur zur Bewertung von CI/CD-Pipelines . .	28
3	Integration- und Delivery-Zeit in Sekunden	47
4	Ergebnistabelle zum AHP	53

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-Ressource-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. Die CEA ist ein IT-Konzept, welches Unternehmen ermöglicht, Systeme aus modularen wiederverwendbaren Software-Komponenten zusammenzusetzen [60]. Laut Martin Henning, Head of New Ventures and Technologies bei der SAP, sind Unternehmen mit diesem Architekturkonzept nicht länger auf „rigide ERP-Software“ angewiesen, sondern vielmehr in der Lage, Geschäftsprozesse auf Grundlage einzelner Software-Bausteine zu digitalisieren [53]. 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 betrachten werden [64]. 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 Entwicklerpraktik. Damit soll sichergestellt werden, dass Änderungen am Code kontinuierlich und zuverlässig in den Produktionssystemen bereitgestellt werden. Dies kann mithilfe einer CI/CD-Pipeline realisiert werden. Mit dieser ist es möglich, die in einer Entwicklungsabteilung anfallenden Softwarebereitstellungsprozesse vollständig zu automatisieren. Empirische Daten belegen, dass der Einsatz dieser Pipelines den Bereitstellungsprozess im Durchschnitt um das Doppelte beschleunigt, während die Fehlerquote um 50 Prozent reduziert wird [31]. Da CI/CD-Pipelines eine kontinuierliche Bereitstellung und Integration modularer Software-Komponenten ermöglichen, sind diese insbesondere für die CEA 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 Services (SAP DTS) unterstützt Kunden bei der Implementierung von ERP-Services auf der Cloud-Plattform SAP Business Technology Platform (SAP BTP). Um ihren Kunden optimale Lösungen anzubieten, sind aktuelle Technologieinnovationen für das SAP DTS von hoher Bedeutung. In diesem Kontext stellt ebenfalls die CEA im Bereich Unternehmenssoftware ein hochrelevantes und zukunftsweisendes Thema dar. Neben der Planung, dem Entwurf und der Erstellung von IT-Services berät das SAP DTS Kunden ebenfalls in der Implementierung von Softwarebereitstellungsprozessen. 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 Unternehmen übertragbar ist. Aus diesem Grund ist in der vorliegenden Arbeit ebenfalls vorgesehen, das Ergebnis in einer abschließenden Handlungsempfehlung einzuordnen und zu differenzieren. 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 BTP 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

Im ersten Teil der Arbeit erfolgt eine Definition 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 dargestellt, wie Composable-Enterprises (CEs) ihre Architektur gestalten müssen, um geschäftlichen Abläufe adäquat in der Unternehmenssoftware abzubilden. Im nachfolgenden Kapitel 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 in der CI/CD-Pipeline abgewickelten Schritte sowie bei der SAP verwendete Tools be-

schrieben. 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.

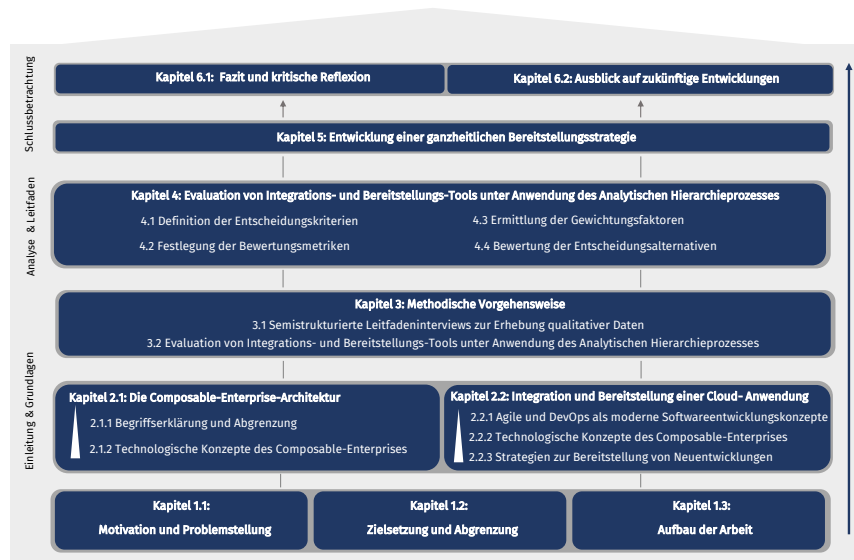


Abbildung 1: Aufbau der Arbeit. Eigene Darstellung.

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 Evaluierung zukünftiger Forschungsansätze.

2 Grundlagen und Begriffserklärungen

2.1 Die Composable-Enterprise-Architektur

2.1.1 Begriffserklärung und Abgrenzung

Nach Ansicht von Steve Denning, Managementberater und Autor der Forbes, stellen Flexibilität, Resilienz und Agilität wesentliche Faktoren dar, welche zur Steigerung der Wettbewerbsfähigkeit von Unternehmen beitragen [49]. Für Analystenhäuser wie Gartner steht fest, dass es technologischer Innovation benötigt, um einhergehende Herausforderungen erfolgreich zu bewältigen und eine kontinuierliche Unternehmenstransformation voranzutreiben. Gartner empfiehlt dabei monolithische und starre Softwarearchitekturen, durch einen modularen Systemaufbau zu ersetzen. In seinen Veröffentlichungen verwendet Gartner für dieses Konzept den Begriff der **Composable-Enterprise-Architektur (CEA)**.

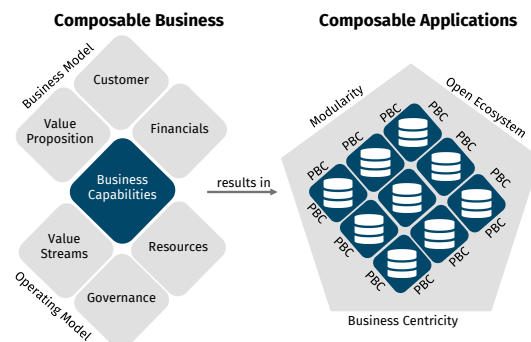


Abbildung 2: Entstehung einer Composable-Enterprise-Architektur. In Anlehnung an Schönstein [74].

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 [54].“

Eine CEA beschreibt somit ein aus mehreren Bausteinen, sog. *Packaged-Business-Capability (PBC)* bestehenden Systemaufbau. PBCs sind vorgefertigte Softwareelemente, welche jeweils eine bestimmte Geschäftsfunktion abdecken (s. Abb. 2). Das

Konzept der PBCs basiert dabei auf den drei Prinzipien der CEA: *Modulare Architektur*, *Offenes Ökosystem* und *Businesszentriertheit* [60]. 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“ [60]. 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 für 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 [54]. 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. Somit sind Unternehmen in der Lage, die Rechenkapazität einzelner ERP-Module flexibel und kosteneffizient an wachsende Geschäftsanforderungen anzupassen [80, S. 7].

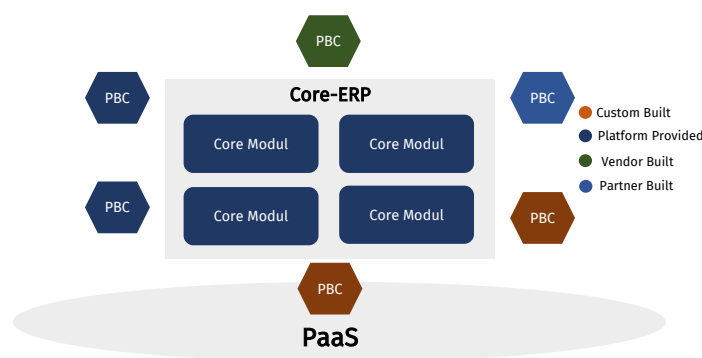


Abbildung 3: Bestandteile eines Composable-ERP-Systems. Eigene Darstellung.

Ein Composable-ERP-System besteht aus einer 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 aufeinander abgestimmt werden [42]. 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 auf Künstlicher Intelligenz (KI) basierenden Kundenanalysefunktion zu erweitern. Um die Integration dieser modularen Komponenten zu erleichtern, stellen ERP-Anbieter auf ihren Plattformen einen zur Bündelung der PBCs verwendeten Marktplatz zur Verfügung [42]. 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 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 [60]. IT-Systeme sind primär auf eine Unterstützung operativer Tätigkeiten ausgerichtet. 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 PBCs intuitiv nutzen und ggf. weiterentwickeln und anpassen können, ohne dabei von IT-Abteilungen abhängig zu sein [60]. 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*: *Microservice*, *API*, *Cloud-native*, *Headless*.

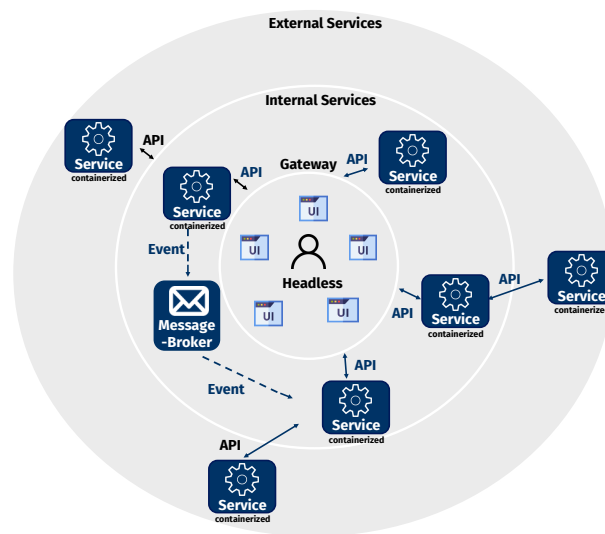


Abbildung 4: Technische Realisierung der Composable-Enterprise-Architektur. Eigene Darstellung.

Der zentrale Einstiegspunkt in eine CE-Anwendung ist das User-Interface. Für die Frontend-Entwicklungen wird dabei i.d.R. 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 [2, S. 4]. Dies gewährleistet eine individuelle Gestaltung der Benutzeroberfläche über verschiedene Kanäle und Plattformen (z. B. Web, Mobile) hinweg. 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 [8, S. 93 ff.]. 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 [3, S. 42]. 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 [5, 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 [7, S. 54]. Alle Komponenten der CEA werden auf einer Cloud-Plattform betrieben (*Cloud-native*) [19, S. 3]. Cloud-Computing ist ein Dienstleistungsmodell, welches Nutzern ermöglicht Ressourcen, wie Speicher, Analyse-Tools oder Software über das Internet von einem Cloud-Anbieter zu beziehen [23, 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 [23, 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)* [23] [23, S. 9]. Bei diesem Computing-Modell wird eine Plattform bereitgestellt, auf welcher Kunden eigene Dienste entwickeln, testen und betreiben können. Ein auf dieser Service-Ebene von der SAP bereitgestelltes Produkt ist die 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

Um eine maßgeschneiderte Composable-ERP-Lösung bereitzustellen, ist es erforderlich, dass Unternehmen das System um eigenentwickelte Microservices erweitern oder bereits bestehende Komponenten modifizieren. Zur Koordination des dabei resultierenden Entwicklungs- und Bereitstellungsprozesses, werden i.d.R. verschiedene Softwareentwicklungskonzepte verwendet. Dabei gibt es das traditionelle Wasserfallmodell, welches eine sequenzielle Abfolge der Projektelemente *Anforderung*, *Design*, *Implementierung*, *Test* und *Betrieb* vorgibt. Im Hinblick auf die mit einer CEA angestrebten Flexibilität weist dieses Entwicklungskonzept jedoch merkbare Beschränkungen auf. Die in dieser Methodik detailliert durchgeführte Vorabplanung, kann insbesondere in umfangreichen Langzeitvorhaben aufgrund unvorhersehbarer Externalitäten selten eingehalten werden [29, S. 5]. Dies resultiert nicht nur in einem Anstieg der Kosten, sondern führt ebenfalls dazu, dass IT-Projekte länger als geplant ausfallen [28, 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) [13, S. 87]. Sprints repräsentieren Durchläufe, welche i.d.R. eine Dauer von ein bis vier Wochen aufweisen. Alle innerhalb des Projektumfangs zu entwickelnden Funktionalitäten werden dabei in einem zentralen Artefakt (*Product Backlog*) festgehalten und von dem Produktverantwortlichen (*Product Owner*) priorisiert [11, S. 196].

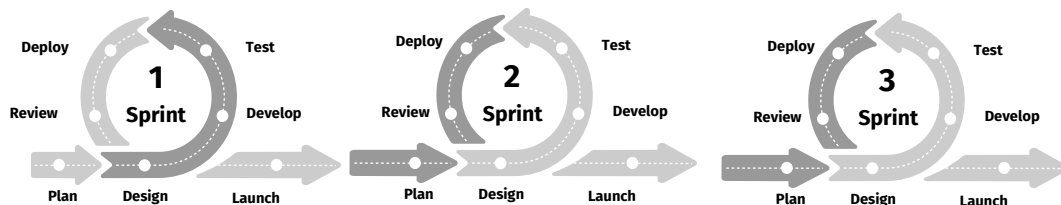


Abbildung 5: Exemplarische Abfolge eines agilen Entwicklungszykluses. In Anlehnung an K&C [32].

Während eines Sprints ist die Fertigstellung eines vor diesem Abschnitt definierten Aufgabenkontingents (*Sprint Backlog*) vorgesehen [21, S. 194]. 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 daraus resultierenden Anforderungen im unmittelbaren Folge-Sprint eingearbeitet werden [12, S. 180 ff.]. 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 [37]. Traditionell erfolgt eine funktionale Trennung der Entwickler- und IT-Betrieber-Teams. Das Problem der letzten Meile beschreibt dabei, dass aufgrund ausbleibender Kooperation dieser Teams der Programmcode nicht auf die Produktivumgebung abgestimmt ist [15, S. 17]. 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 [14, S. 1]. So geht aus der von Avasant veröffentlichten Studie *IT-Spending And Staffing Benchmarks* hervor, dass durchschnittlich 75 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 [59, S. 6]. Abhilfe schaffen kann das in der Literatur als **Development & Operations (DevOps)** bekannte Aufbrechen organisatorischer Silos zwischen Entwicklung und dem IT-Betrieb [14, 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 CAM: *Culture (Kultur)*, *Automation (Automatisierung)* und *Measurement (Messung)* [14, S. 5]. Dabei gilt *Kultur* als das wohl wesentlichste DevOps-Erfolgselement. Diese bezweckt eine Kollabora-

tionsmentalität, 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 [14, 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 [14, S. 7]. Dieser beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [27, S. 141]. Dabei soll die mit DevOps angestrebte Verschmelzung der Entwicklungs- und Betriebsteams, die Automatisierung von Prozessen sowie die kontinuierliche Integration und Bereitstellung von IT-Services zu einer erheblichen Reduzierung dieser Metrik führen. Martin Fowler, Chief Scientist des Softwaredesignunternehmens ThoughtWorks, 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 [51].

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 beitragen soll. Abhilfe schaffen soll dabei eine sog. CI/CD-Pipeline, welche alle Schritte von Integration des Codes bis Bereitstellung der Software automatisiert. Hauptaugenmerk liegt dabei auf einer zuverlässigen und kontinuierlichen Bereitstellung von Software [30, S. 471]. 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 wer-

den. Das frühzeitige Integrieren von Code soll dabei zu einer unmittelbaren und zuverlässigen Fehlererkennung innerhalb des Entwicklungsvorhabens beitragen [30, S. 471].

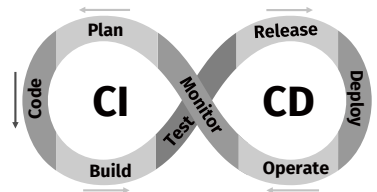


Abbildung 6: Aktivitäten im CI/CD-Prozess. In Anlehnung an Synopsys [88].

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 Produktivsystems abgestimmt ist, sollte zu jedem Zeitpunkt das Know-how der Betriebsteams einbezogen werden [14, S. 16]. Nach erfolgreichem Entwurf zu implementierender Anwendungs-Features beginnt die Entwicklung der IT-Services. Arbeiten hierbei mehrere Entwickler parallel an demselben IT-Service, wird der entsprechende Quellcode in Versionsverwaltungssysteme sog. *Repositories* wie Github oder Bitbucket ausgelagert.

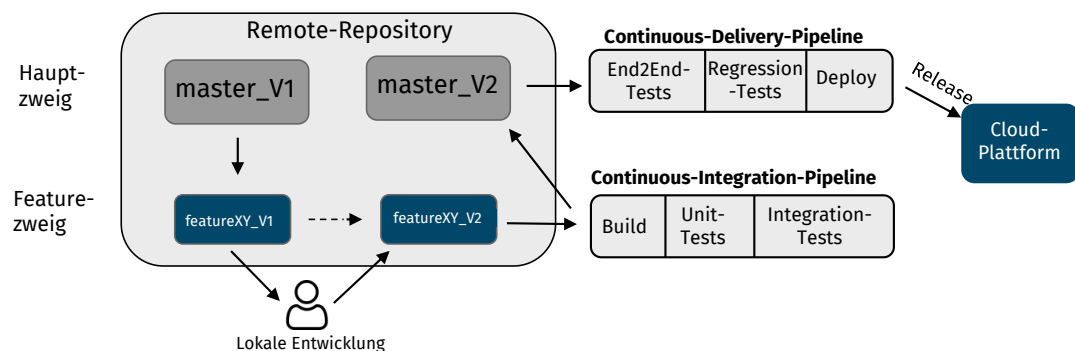


Abbildung 7: Integration der CI/CD-Pipeline mit dem Versionskontrollsystem. Eigene Darstellung.

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 [6, S. 1]. 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) [26, S. 337]. 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 Code-Basis zu verursachen. Nach Fertigstellung der Funktionalitäten sollte der um die Features erweiterte Quellcode so schnell wie möglich in den Hauptzweig integriert werden [26, S. 337]. 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 Bereitstellungsautomatisierungs-Tool, welches auf einer virtuellen Maschine oder in einer containerisierten Computing-Umgebung betrieben wird [43]. 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 [20, S. 737]. Nach Ablauf der Build-Workflows erfolgt eine automatische Abwicklung des Validierungsprozesses (*Smoke-Tests*). Damit soll sichergestellt werden, dass zu jeder Zeit ein rudimentär getesteter Code bereitsteht und grundlegende Funktionalitäten sowie Schnittstellen erwartungsgemäß ausgeführt werden [14, S. 19]. Die in diesem Schritt abgewickelten Tests leiten sich dabei aus der *Definition of Done (DoD)* ab. Die DoD ist eine in der Planungsphase festgelegte Anforderungsspezifikation, deren Erfüllung als notwendige Voraussetzung für den Abschluss eines Features gilt. Somit sind Entwickler dazu angehalten, für jedes implementierte Feature einen der DoD entsprechenden Tests zu entwerfen (*Test Driven Development*) [84]. Der in dem CI-Prozess bereitgestellte Code wird da-

bei 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*. Unit-Tests befinden sich dabei auf unterster Hierarchieebene der Test-Pyramide (s. Abb. 8).

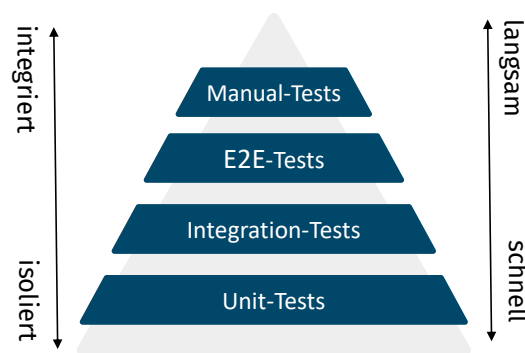


Abbildung 8: Hierarchische Darstellung von Softwaretests.
In Anlehnung an Paspelava [66].

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 [17, S. 177]. Der Zweck der Unit-Tests besteht dabei in einer von externen Einflüssen und Daten unabhängigen Überprüfung der einzelnen Komponenten. Um bei der Bereitstellung neuer Funktionalitäten ebenfalls das Zusammenspiel verschiedener Komponenten zu überprüfen, werden *Integration-Tests* durchgeführt [17, S. 177]. Bei diesen Tests können Aspekte, wie der Austausch eines Nachrichtenmodells zweier Web-Services oder das Response-Objekt einer Datenbankabfrage untersucht werden. 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 [14, S. 19 ff.]. 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 zwischengeschaltet [14, S. 19 ff.]. 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 [71]. 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 [14, S. 20]. Darüber hinaus werden in der Staging Area ebenfalls in der Test-Pyramide (s. Abb. 8) höher positionierte, also rechenintensivere Tests ausgeführt [39]. 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 [39]. Für kritische Systeme werden während des Delivery-Prozesses ebenfalls *Regression-Tests* vorgenommen. Diese umfassen ein erneutes Testen bereits ausgelieferter Softwarekomponenten [10, S. 15 ff.]. Regression-Tests können dabei in Form von Unit-, Integration- sowie E2E-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 [57]. Da diese Tests nicht automatisiert durchgeführt werden können, muss der nächste Schritt der CD-Pipeline nach erfolgreicher Validierung manuell angestoßen werden. Im Anschluss an die funktionalen Tests werden i.d.R. verschiedene Code-Analysen abgewickelt. Hierbei werden Metriken, wie die prozentuale Testabdeckung oder Schwachstellen verwendeter Code-Patterns überprüft [22, S. 146]. Nach Durchführung der Code-Analysen wird das überprüfte Artefakt auf das Produktionssystem der 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* [86][89][58]. 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.

Zur Automatisierung der CI/CD-Prozesse werden von dem SAP DTS 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 [90]. Eine weitere bei der SAP empfohlene CI/CD-Alternative ist das Open-Source-Tool *Jenkins*. Im Gegensatz zum templatebasierten SAP CI/CD, muss der Bereitstellungsworkflow 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 [4, S. 266]. 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 zu den bei SAP CI/CD bereitgestellten Templates sind diese jedoch hoch konfigurierbar. Ein externes ebenfalls von der SAP vorgeschlagenes CI/CD-Werkzeug ist *Azure Pipelines*. Azure Pipelines ist ein von Microsoft entwickeltes Tool, 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 [87].

2.2.3 Strategien zur Bereitstellung von Neuentwicklungen

Nachdem das Artefakt auf einer 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 IT-Services in die Produktionsumgebung bereitgestellt werden 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 [81]. 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 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 von dem Lastverteilungsservice (*Load-Balancer*) unmittelbar auf diese umgeleitet werden. Somit können längere Ausfallszeiten umgangen werden [24, S. 1083].

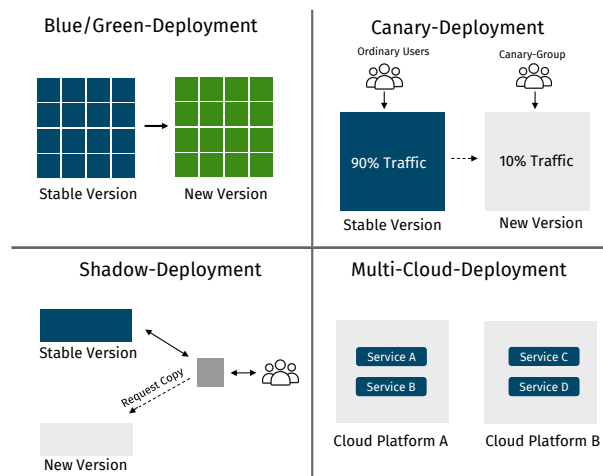


Abbildung 9: Strategien zur Bereitstellung von Software.
In Anlehnung an Ugochi [81].

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 Anwendungen überprüft werden. Bevor die neue Anwendungsversion der gesamten Nutzerbasis zur Verfügung gestellt wird, kann diese sukzessive und schrittweise an eine zunehmende Anwenderanzahl ausgerollt werden [81]. Dies ermöglicht eine umfassende Überwachung und Kontrolle bezüglich potenziell im Produktionssystem auftretender Probleme. 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 an die aktuelle Version der Instanz übermittelt und von dieser 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 [81]. Unabhängig vom Zeitpunkt, zu welchem ein IT-Service für die Nutzer bereitgestellt wird, erfolgt der Anwendungsbetrieb dabei entweder im *Single-* bzw. *Multi-Cloud-Deployment*. Im Single-Cloud-Deployment wird eine Anwendung stets auf einer einzigen Cloud-Plattform betrieben. Im Gegensatz dazu wird das *Multi-Cloud-Deployment* verwendet, um ERP-Dienste in einer verteilten Computing-Umgebung auszuführen. Dabei werden die IT-Services 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 [85].

3 Methodische Vorgehensweise

3.1 Semistrukturierte Leitfadeninterviews zur Erhebung qualitativer Daten

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. 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 [18, S. 33 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* [18, S. 84 ff.]. Strukturierte Interviews 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 [1, S. 421 ff.]. Im anderen Extrem der unstrukturierten Interviews erfolgt lediglich eine Bestimmung des Gesprächsthemas, jedoch werden vorab keine expliziten Fragen festgelegt [1, S. 441 ff.]. 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. Das liegt insbesondere an dem starren Erhebungsdesign dieser Interviewform. Angesichts der in dieser Arbeit angestrebten Erschließung unerforschter Themengebiete bietet eine Vorabfestlegung der Fragen nicht genügend Flexibilität, um auf neu aufkommende Aspekte innerhalb des Gesprächs angemessen zu reagieren. Deshalb werden im Rahmen dieser Arbeit semistrukturierte Interviews durchgeführt. Diese Interviewform realisiert eine Leitfadenstruktur, welche eine vordefinierte Fragegestaltung vorgibt, jedoch im Verlauf des Gesprächs gleichzeitig ein hohes Maß an Flexibilität bietet. Ergeben sich während eines Expertengesprächs neue Aspekte, können diese mit unmittelbaren Ad-hoc-Fragen aufgegriffen werden. Um die Interviews auszuwerten, muss eine Transkription der Expertengespräche erfolgen [1, S. 179]. 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 [1, S. 663]. 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 [1, S. 663]. 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

Zur Analyse und Bewertung der CI/CD-Tools wird das AHP-Verfahren angewandt. 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. Das zugrundeliegende Rahmenwerk basiert dabei auf der Prämisse einer divergierenden Wichtigkeit der unterschiedlichen Entscheidungskriterien. Diese Methode erweist sich also als besonders geeignet, wenn es erforderlich ist, eine Bewertung von Entscheidungsalternativen auf Präferenzen verschiedener Stakeholder abzustimmen [25, 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).

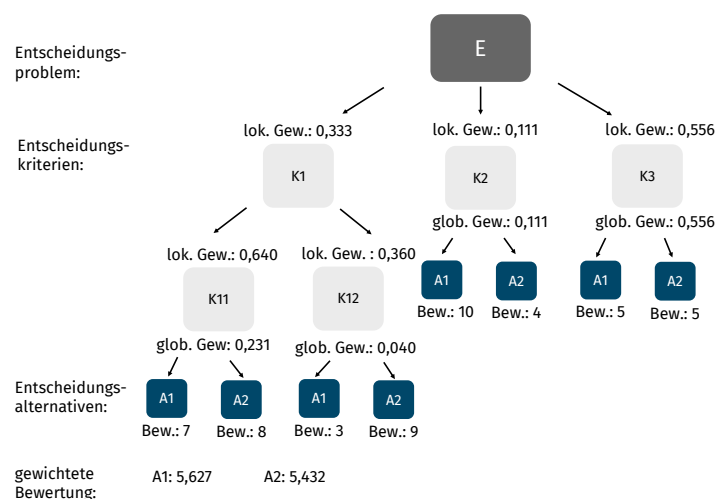


Abbildung 10: Exemplarische Darstellung der hierarchischen Entscheidungsstruktur im AHP. Eigene Darstellung.

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 [25, S. 86]. 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 [25, 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 _j)	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 [16, S. 14341]. Um die globale Gewichtung (P_{ki}) zu ermitteln, wird die Gewichtung jedes Subkriteriums mit den Gewichtungen der übergeordneten Kriterien multipliziert :

$$P_{Ki} = W_{K1} \cdot W_{K2} \cdot W_{K3} \cdot W_{Kn}$$

In der Literatur wird vorgesehen, dass auf unterster Hierarchieebene ebenfalls eine Gewichtung der Entscheidungsalternativen vorgenommen wird, um eine Bewertung durchzuführen [25, S. 88]. 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 abgewinkelte 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-Tools unter Anwendung des Analytischen Hierarchieprozesses

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 von Spezialisten verschiedener Fachbereiche konsolidiert 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 Elemente, welche die in vergangenen Kundenprojekten hervorgegangenen Anforderungen an eine CI/CD-Pipeline widerspiegeln. Neben den in externen Projekten anfallenden Anforderungen werden bei der Kriterienbildung ebenfalls interne, von der SAP zur Bereitstellung von Standardsoftware festgelegte Bestimmungen berücksichtigt. Obwohl Kunden bei der Entwicklung eigener Services nicht zur Einhaltung dieser Richtlinien verpflichtet sind, erlaubt eine Berücksichtigung dieser Aspekte die Festlegung eines Optimalzustands. Die bei der induktiven Kodierung erhobenen Kategorien werden anschließend ebenfalls als Entscheidungskriterien im AHP-Verfahren wiederverwendet. Die mit dieser Vorgehensweise erhobenen Entscheidungsalternativen sind der Tabelle 2 zu entnehmen. 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 IT-Services auf der Cloud-Plattform bereitzustellen [98, Z. 72 ff.]. Angesichts der Vielfältigkeit des Entscheidungskriteriums Funktionalität wird eine Untergliederung in verschiedene Subkriterien vorgenommen.










	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

Tabelle 2: AHP-Entscheidungsstruktur zur Bewertung von CI/CD-Pipelines.
Eigene Darstellung.

In Kriterium K1.1 wird dabei die Unterstützung automatisierter Unit-Tests evaluiert [98, 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 kontinuierlich und automatisiert durchgeführte Prüfung [92, Z. 65 ff.]. Dies umfasst eine Abwicklung von Unit-, Integration-, E2E-Test sowie Regression-Tests für Backend sowie Frontend. Im Rahmen der SAP-Produktstandards werden dabei verschiedene Test-Tools empfohlen. Hinsichtlich der Entwicklung von SAP-CAP-Node-Anwendungen wird die Durchführung von Unit-Tests mittels Jest bzw. Mocha und

Integration-Tests mittels Newmann vorgeschlagen. Für die Programmierung mit SAP UI5 sind hingegen Unit-Tests mittels Q-Unit, Integration-Tests mittels OPA5 und E2E-Tests mittels WDI5 vorgesehen [92, 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 [98, 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 Code-Analysen, 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 [97, 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-Pipeline-Tools evaluiert [98, Z. 71 ff.]. Sowohl für SAP-UI5- als auch SAP-CAP-Node-Anwendungen wird bei der Kompilierung das *Multi-Target-Application-Konzept (MTA-Konzept)* verwendet. 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. Eine MTA besitzt dabei einen sog. *MTA-Deskriptor*, welcher die zur Kompilierung benötigten Konfigurationen enthält. Diese mit dem MTA-Konzept vorgesehene Modul-Bündelung ermöglicht somit, dass komplexe aus verschiedenen Komponenten bestehende Microservices in einer einzigen Datei beschrieben werden können [62]. Da für das Kompilieren von MTAs das Build-Tool Make benötigt wird, wird evaluiert, ob dieses durch die Pipeline-Tools unterstützt wird. Laut Experte 1 ist weiterhin essenziell, dass Artefakt-Repositories durch die Pipeline-Tools unterstützt werden [98, Z. 15 ff.]. Eine MTA kann von verschiedenen externen Ressourcen abhängig

sein. Diese sind in einem Artefakt-Repository verwaltete Komponenten, welche bei der Entwicklung neuer Microservices wiederverwendet werden können [98, Z. 40 ff.]. Diese Komponenten werden während des Build-Prozesses von der CI/CD-Pipeline aus den Artefakt-Repositories geladen und kompiliert. Aus diesem Grund ist es von bedeutendem Mehrwert, wenn CI/CD-Tools entweder über ein integriertes Artefakt-Repository verfügen oder die Möglichkeit der Einbindung externer Versionsverwaltungssysteme besteht. Ein weiterer für CEs essenzieller Build-Aspekt ist die Unterstützung von Kompilierungs-Tools für Docker-Container. Docker-Container sind portable und isolierte Laufzeitumgebungen, welche alle notwendigen Abhängigkeiten und Ressourcen einer Anwendung beinhalten. Somit können virtualisierte Umgebungen mit benötigten Frameworks und Tools bereitgestellt werden, ohne dabei eine gesamte Infrastruktur manuell konfigurieren zu müssen. [33]. In Kriterium K1.4 werden die *Deploy- und Release-Funktionalitäten* der CI/CD-Tools untersucht [98, 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. 12). 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)[96, Z. 65 ff.]. In Kategorie K1.5 wird die *Monitoring-Funktionalität* der verschiedenen CI/CD-Pipelines untersucht [96, 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 [96, 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 insbesondere für externe Kundenprojekte evaluiert werden, ist innerhalb dieses Kriteriums ebenfalls eine Betrachtung von externen Open-Source-Tools vorgesehen. Zur Eingrenzung des Bewertungsumfangs, fokussiert

sich die Open-Source-Untersuchung ausschließlich auf das Kibana-Dashboard, welches aufgrund seiner weiten Verbreitung bei Kunden von besonderer Relevanz ist [96, Z. 74 ff.]. In Kategorie 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 [98, Z. 89 ff.]. Damit können bestimmte Ereignisse, wie Push-Mitteilungen bei Code-Änderungen automatisiert an die CI/CD-Pipeline übermittelt werden, was in einer unmittelbaren Auslösung des Integrations- bzw. Bereitstellungs-Workflows der CI/CD-Pipeline resultiert. 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 [93, Z. 95 ff.]. In Kriterium K2.2 werden die *Integrationsmöglichkeiten von Entwicklungsumgebungen* untersucht [98, 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 [93, 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 [93, 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 In-

tegrationsfähigkeit von Planungssoftware [93, Z. 96 ff.]. In Kriterium K3 erfolgt die Evaluation der **Kosten** [96, Z. 42 ff.]. Mit diesem Entscheidungskriterium werden die durch die CI/CD-Pipelines verursachten *Total Cost of Ownership (TCO)* evaluiert. TCO beschreiben den Geldbetrag, welchen ein Unternehmen während des gesamten Lebenszykluses einer Investition, also von Beschaffung bis zur vollständigen Entsorgung, aufbringen muss [9, S. 4]. 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 [98, 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 bedeutenden Mehrwert. Die vertikale Skalierung bezieht sich auf die Erhöhung der Ressourcen einer einzigen Pipeline-Instanz. Laut Experte 1 kann die CI/CD-Pipeline so dynamisch an die sich ändernden Leistungsanforderungen angepasst werden [98, Z. 74 ff.]. In Kriterium K5 wird die **Performance** der Pipelines verglichen [96, 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. 14). 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 [98, 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 [98, 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. Da mit diesen ebenfalls nicht im Standard verfügbare Funktionalitäten in die Pipeline integriert werden können, besteht für Entwicklungsabteilungen 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 [96, 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. Somit soll der Wissenstransfer und das Verständnis der involvierten Technologien und Prozesse innerhalb der Entwicklungsteams gefördert werden. 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 [96, Z. 46 ff.]. Die Qualität des Community-Supports hängt dabei i.d.R. von dem Kontributionsmaß der Pipeline-Tools 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 [76]. In dem Kriterium K8 wird die **Sicherheit** der CI/CD-Pipelines untersucht [98,

Z. 79 ff.]. Um unerwünschte Zugriffe zu vermeiden, sollte die CI/CD-Pipeline ein Authentifizierungs- und Autorisierungskonzept unterstützen. Besonders vorteilhaft ist dabei die Einbindung zentralisierter Drittanbieter, wie der SAP Identity Provider oder GitHub. Ein weiterer 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 das Aufsetzen und Einrichten des Pipeline-Systems nicht selbst übernommen werden muss, sondern unmittelbar als Service bereitgestellt wird [98, Z. 67 ff.]. Auch der für die Implementierung und Konfiguration der Pipelines benötigte Aufwand sollte so gering wie möglich sein (*intuitive Bedienbarkeit*) [98, 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. Für qualitative, in dem Entscheidungs-Framework zu betrachtende Kriterien wird eine gewichtende Bewertung vorgenommen:

- **0 Punkte:** Ausschließlich Nachteile
- **1 Punkt:** Nachteile überwiegen Vorteilen
- **2 Punkte:** Vorteile und Nachteile gleichgewichtig

- **3 Punkte:** Vorteile überwiegen Nachteilen
- **4 Punkte:** Ausschließlich Vorteile

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 Anforderungen eingegangen werden. Aufgrund des qualitativen Evaluationsdesigns wird diese Metrik für die Kriterien *Funktionalität (K1)*, *Integrationsmöglichkeiten (K2)*, *Skalierbarkeit (K4)*, *Flexibilität (K5)*, *Administrativer Support (K7.1)*, *Sicherheit (K8)* und *Benutzerfreundlichkeit (K9)* angewendet. Da für die Pipelines unterschiedliche Preismetriken festgelegt sind, kann das Kriterium *Kosten (K3)* ebenfalls ausschließlich anhand der qualitativen Metrik bewertet werden. Für das quantitative Kriterium *Performance (K5)* wird eine divergierende Metrik definiert:

- **0 Punkte:** 75 Prozent oder mehr über dem niedrigsten Wert
- **1 Punkt:** 50 Prozent bis 75 Prozent über dem niedrigsten Wert
- **2 Punkte:** 25 Prozent bis 50 Prozent über dem niedrigsten Wert
- **3 Punkte:** 0 Prozent bis 25 Prozent über dem niedrigsten Wert
- **4 Punkte:** Niedrigster Wert

Mithilfe dieser Metrik lassen sich die relativen Integration- bzw. Delivery-Zeiten 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 Evaluationsdesign 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 nach der zuvor definierten qualitativen Metrik. So werden vier Punkte für die höchste Blog-Post-Anzahl vergeben. Für die restlichen Bewertungen wird invers nach den in der qualitativen Metrik 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. In Abb. 11 werden die lokalen und globalen Durchschnittsgewichtungen der AHP-Entscheidungskriterien dargestellt:










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 11: Gewichtungsfaktoren der AHP-Entscheidungskriterien. Eigene Darstellung.

Die Experten legen dabei auf unterschiedliche Aspekte Wert. So ist für den Softwarearchitekten die von den Pipelines bereitgestellte Funktionalität von hoher Bedeutung. Innerhalb dieses Kriteriums besonders ausschlaggebend ist für diesen Experten

dabei die Unterstützung verschiedener Test-Frameworks. Nach Auffassung des Architekten besteht zwar die Möglichkeit, Tests manuell und unabhängig von einer Pipeline auszuführen. Allerdings gestaltet sich dabei die Überwachung der Einhaltung dieser Testrichtlinien als sehr herausfordernd [99, Z. 18 ff.]. Bei einer CEA ist es üblich, dass eine hohe Bandbreite verschiedener Programmier-Frameworks eingesetzt wird [99, Z. 13 ff.]. Aus diesem Grund ist es für den Architekten ebenfalls essenziell, dass Pipelines unterschiedliche Build-Tools unterstützen. Für den Softwarearchitekten weniger wichtig ist hingegen der Installations- und Wartungsaufwand. Dies ist darauf zurückzuführen, dass die Pipeline-Systeme i.d.R. einmalig aufgesetzt werden und die Einrichtung somit keine kontinuierlich anfallende Tätigkeit darstellt. Sowohl für den Fullstack- als auch die Test-Entwickler stellt die Unterstützung automatisierter Tests einen signifikanten Aspekt dar [91, Z. 5 ff.]. Da die Entwickler ein möglichst zeitnahes Feedback auf Code-Änderungen erhalten möchten, ist bei der Abwicklung dieser Tests insbesondere die Integration-Zeit, also das Intervall der Konsolidierung von Code-Änderungen in den Hauptzweig, essenziell [92, Z. 23 ff.]. Für die Entwickler weniger wichtig ist in diesem Kontext die Delivery-Zeit, also die Dauer des Ausrollens von IT-Services auf das Produktionssystem. Dies ist darauf zurückzuführen, dass sich die Geschwindigkeit dieses Prozesses nicht unmittelbar auf die Produktivität und Effizienz der Entwickler auswirkt. Kosten sind die Entwickler ebenfalls von geringer Relevanz, da die Experten während der operativen Tätigkeit kaum Berührungspunkte mit diesem Aspekt besitzen. Eine höhere Bedeutung hat für die Entwickler hingegen der Community-Support [94, 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 [95, Z. 5 ff.]. Weniger wichtig sind für den Product Manager hingegen die Integrationsmöglichkeiten von Planungssoftware sowie Entwicklungsumgebung. Dies begründet der Experte damit, dass derartige Integrationen in der tatsächlichen Anwendungspraxis wenig Verwendung finden und die Wahl eines Pipeline-Tools daher durch diese Aspekte nicht tangiert wird. Weiterhin

erachtet der Product Manager 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 [95, 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 wie vorimplementierte Skripte, Test-Frameworks sowie Laufzeitumgebungen 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 [93, 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 [47][65]. Dieser gewährleistet ein automatisiertes Ausführen von Pipelines abhängiger Microservices. Dabei können sämtliche Tests der Anwendungen, welche den neuen Service konsumieren, erneut durchgeführt und validiert werden. In SAP CI/CD können alle Tests, mit Ausnahme der Backend-Integration-Tests mittels Newmann und der 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 [97, Z. 40 ff.]. Mit dem SAP CI/CD-Tool sind ausschließlich statische Code-Analysen mittels Lint bzw. SonarQube möglich [98, Z. 50 ff.]. Neben der fehlenden Unterstützung von Performance-Tests birgt insbesondere die Inkompatibilität von Sicherheits-Tools für eine 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 [97, Z. 37 ff.]. Somit kann das SAP CI/CD nicht von Kunden verwendet werden, welche sich an dem Sicherheitsstandard der SAP orientieren. Aus diesem Grund ergibt sich eine Bewertung von einem Punkt für SAP CI/CD (Nachteile überwiegen) und vier Punkten für Jenkins bzw. Azure Pipelines (ausschließlich Vorteile). In Kriterium K1.3 wird die *Build-Funktionalität* der CI/CD-Tools evaluiert. Mit Project Piper wird dabei das für SAP UI5 sowie SAP CAP Node benötigte MTA-Build-Tool zur Verfügung gestellt. Weiterhin wird durch die Programmbibliothek ein Build-Tool für Docker-Container sowie eine Funktion zur Einbindung des von der SAP bereitgestellten Artefakt-Repositorys Artefactory bereitgestellt [41]. SAP CI/CD unterstützt kein Docker-Workflow sowie Artefakt-Repository [70]. 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 eines Services kann zu einer früheren im Artefakt-Repository abgelegten Version zurückgekehrt werden. Somit wird das Risiko von Ausfällen und Unterbrechungen im Geschäftsbetrieb minimiert. Da SAP CI/CD 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.

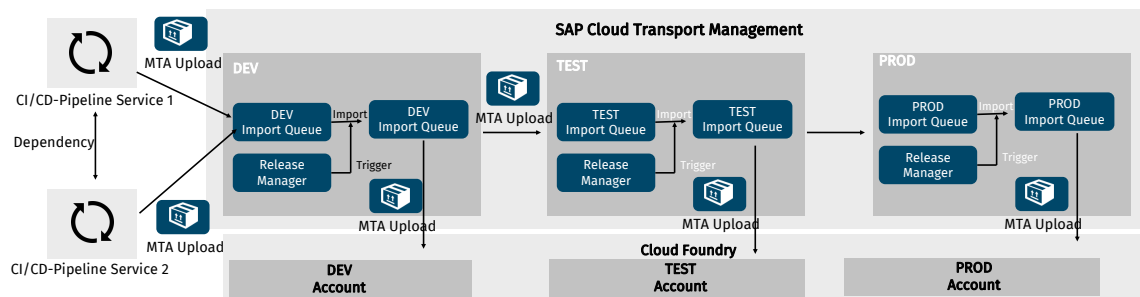


Abbildung 12: SAP Cloud Transport Management. In Anlehnung an Stevens [77].

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 eine CEA von hoher Bedeutung ist [96, 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. 12). Weiterhin wird von den Pipelines neben dem Multi-Cloud- ebenfalls das Blue/Green-Deployment unterstützt [44]. Da das Blue/Green-Deployment eine hohe Flexibilität und Agilität im Bereitstellungsprozess gewährleistet, stellt dieses für CEs einen hohen Mehrwert dar. Mit dieser Deployment-Strategie wird neben der zu installierenden neuen Version ebenfalls die stabile Anwendung betrieben. Bei erfolgreicher Inbetriebnahme kann der Datenverkehr unmittelbar auf die neue Instanz umgeleitet werden, wobei Unterbrechungszeiten vermieden werden können. Dies spielt insbesondere im Kontext von Composable-ERP-Systemen, in welchem kritische Prozesse, wie die Überwachung von Zahlungsströmen abgewickelt werden, eine bedeutende Rolle. Während alle CI/CD-Tools eine Cloud-Foundry-, CTM-, Multi-Cloud- 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 [96, 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. 13). 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 [34]. Der SAP CI/CD-

Service unterstützt ausschließlich eine Protokollierung der Build-Logs einzelner Pipelines.

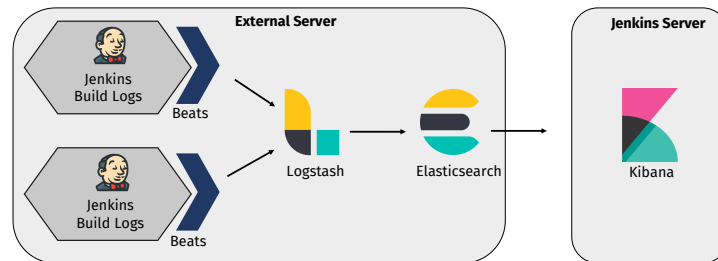


Abbildung 13: Pipeline-Monitoring mit Kibana und Jenkins. In Anlehnung an Atta [34]

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 [96, Z. 38 ff.]. Während für Azure Pipelines vier Punkte vergeben werden (ausschließlich Vorteile), erhält Jenkins aufgrund des hohen Einrichtungsaufwands für das Kibana-Dashboard eine Bewertung von drei Punkten (Vorteile überwiegen). Für SAP CI/CD 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. 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-Branche erst nach erfolgreicher Abwicklung aller Tests in den Hauptzweig integriert werden [98, 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 Webhook-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) [98, 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)*). Das SAP CI/CD unterstützt keine Integration der zu evaluierenden Entwicklungsumgebungen [98, 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 [93, Z. 101 ff.]. Demnach ist es Projektmanagern nicht möglich, den Build-Status verschiedener Backlog-Items zu überwachen. Aus diesem Grund ergibt sich eine Bewertung von vier Punkten für Azure Pipelines sowie Jenkins bzw. null Punkte für das SAP CI/CD. In Kriterium K3 werden die **Kosten** der CI/CD-Tools evaluiert. Für SAP CI/CD werden Gebühren von einem Euro pro Build-Stunde fällig [98, Z. 107 ff.]. Azure Pipelines berechnet hingegen unabhängig von der Nutzungszeit 40 Euro pro Pipeline [35]. 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, fallen

zunächst Investitionskosten für Hardware an. Weiterhin müssen für den On-Premise-Server laufende Betriebskosten, wie Ausgaben für Energie, Reparaturleistungen und Personal zur Wartung der IT-Infrastruktur berücksichtigt werden. Untersuchungsergebnisse des IT-Beratungshauses ExecuTech zeigen, dass On-Premise-Systeme aufgrund hoher Investitionskosten insbesondere bei einer kurzfristigen Betrachtung teurer sind [78]. 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 abzubauen. 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 [48]. 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 [82]. 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 jedoch insbesondere in einem volatilen Geschäftsumfeld eine signifikante Rolle. Als eines der größten CEs stellt der Streaminganbieter Netflix über 1000 Microser-

vices 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 [79][63]. 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) und Jenkins sowie 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. 14).

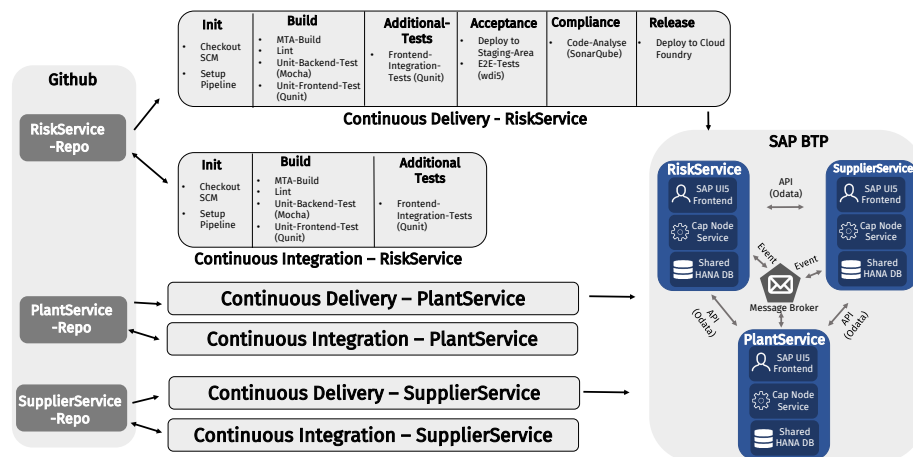


Abbildung 14: CEA-Szenario für Performance-Tests. Eigene Darstellung.

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 SonarQube-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 das 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. 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 21). Die Ergebnisse der Tests zeigen, dass für den vollständigen Durchlauf der CI-Prozesse aller Microservices für Azure Pipelines 600 Sekunden, Jenkins 1134 Sekunden und SAP CI/CD 1243 Sekunden benötigt werden (s. Tab. 3). 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 überdurchschnittliche Leistungsfähigkeit von Azure Pipelines lässt sich auf verschiedene Faktoren zurückführen. Zum einen erfolgt der Betrieb der Pipeline-Systeme unter Einsatz modernster Hardwaretechnologien. Darüber hinaus werden durch Azure Pipelines ebenfalls Mechanismen wie *Parallel Builds* und *Caching* im Standard unterstützt. Während bei Parallel-Builds das Ausführen gleichzeitiger Pipeline-Schritte ermöglicht wird, gewährleistet Caching, dass während des CI/CD-Workflows heruntergeladene Ressourcen für zukünftige Pipeline-Durchläufe zwischengespeichert werden. Dies führt zur erheblichen Beschleunigung der Integration- bzw. Delivery-Prozesse.



		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 3: Integration- und Delivery-Zeit in Sekunden. Eigene Darstellung.

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. 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 [99, Z. 13 ff.]. So sollte die CI/CD-Pipeline verschiedene Programmier-Frameworks, Test-Frameworks und Cloud-Plattformen unterstützen. Damit werden bei Jenkins mehr als 1800 Plug-ins bereitgestellt, mit welchen über den Standard hinausgehende Funktionalitäten integriert werden können [56]. Während mit Azure Pipelines 1400 Plug-ins bereitgestellt werden, besteht diese Möglichkeit bei SAP CI/CD nicht [36]. Weiterhin sollten die CI/CD-Tools, um schnell neue Pipelines implementieren zu können, einen flexiblen

modularen Aufbau ermöglichen. Bei Azure Pipelines und Jenkins können Shared-Librarys, wie Project Piper verwendet werden (s. Abb. 15).

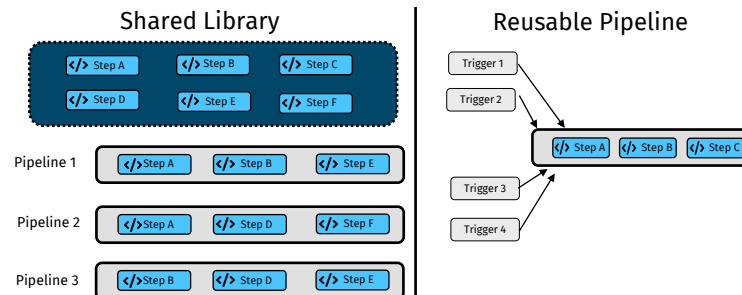


Abbildung 15: Modularer Aufbau einer CI/CD-Pipeline. In Anlehnung an Codefresh [46].

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. Das SAP CI/CD stellt ebenfalls eine Shared-Library bereit. So werden die templatebasierten Schritte als standardisierte Elemente einer CI/CD-Pipeline ausgeliefert. Im Gegensatz zu Azure Pipelines und Jenkins gewährleistet dieses Tool jedoch lediglich die Integration der von dem SAP CI/CD bereitgestellten Bibliothek, wobei die Möglichkeit der Einbindung benutzerdefinierter Librarys nicht besteht. Die Wiederverwendung ganzer Pipelines für unterschiedliche Microservices ist ebenfalls nicht möglich. Während der SAP CI/CD-Service das Shared-Library-Konzept im begrenzten Maße 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, wer-

den 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). Dazu gehört der für Kunden bereitgestellte Premium-Support [83]. Bei diesem 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 sowie der 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 [98, 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 [94, 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 112.000 Posts zur Jenkins-Pipeline veröffentlicht [68]. 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 besteht. So sind für Azure Pipelines 25.000 Post und für SAP CI/CD lediglich 26 Einträge auf Stack-Overflow veröffentlicht [69][67]. Gemäß der in Kapitel 4.2 festgelegten Metrik wird eine Bewertung von vier Punkten für Jenkins (höchste Blog-Post-Anzahl) bzw. von null Punkten für Azure Pipelines und SAP CI/CD vergeben (75 Prozent oder mehr unter der höchsten Blog-Post-Anzahl). 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, Bit-Bucket oder SAP-Single-Sign-On (SAP-SSO) einbinden [55][73][38]. 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 [72]. Azure Pipelines und SAP CI/CD unterstützen ebenfalls Authentifizierungsmöglichkeiten. Die Implementierung eines Rollenkonzeptes kann bei SAP CI/CD 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. Ein ähnlicher Aspekt ergibt

sich in der Systemsicherheit. 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 dieses nicht so gut ausgebildet ist, wie bei den Cloud-Anbietern. Im On-Premise bietet sich hingegen 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 das SAP CI/CD 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 kann. 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 das SAP CI/CD 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 Jenkins eine Bewertung von einem Punkt vergeben (Nachteile überwiegen Vorteilen).










		Globale Gewichtung		Azure Pipelines		Jenkins		SAP CI/CD	
B: Bewertung gB: gewichtete Bewertung		pro hundert		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 Planungssoftware	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	0,0000	0	0,0000	
	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	0	0,0000	4	0,0000	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	
				4,0117		2,2947		1,7349	

Tabelle 4: Ergebnistabelle zum AHP. Eigene Darstellung.

Unter Berücksichtigung der zu untersuchenden Fragestellung ergibt sich eine Gesamtbewertung von 4,0117 für Azure Pipelines, 2,2947 für Jenkins und 1,7349 für SAP CI/CD (s. Tab. 4). 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

Um in einem von intensivem Wettbewerb geprägten Geschäftsumfeld zu bestehen, ist es für Unternehmen von entscheidender Bedeutung, effiziente und agile Bereitstellungungsverfahren zu implementieren. Infolgedessen verwenden 62 Prozent aller Unternehmensentwickler CI/CD-Tools innerhalb ihrer Bereitstellungsprozesse. Das CI/CD-Verfahren bietet dabei insbesondere für CEs erhebliche Vorteile. Obwohl einzelne Module einer CEA grundsätzlich isoliert voneinander betrieben werden, können über Schnittstellen Abhängigkeiten entstehen. Insbesondere bei der Zusammenarbeit vieler Entwickler kann die Implementierung und Integration neuer IT-Services deshalb 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, empfiehlt sich eine Automatisierung korrespondierender CI-Prozesse. Dabei wird der lokale Quellcode 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*), ermöglicht die CI/CD-Pipeline, dass Tests bereits während der Entwicklung durchgeführt werden (*Shift-Left*). Somit erlangen Entwickler kontinuierliches Feedback und können Services optimieren, bis diese den Produktstandards des Unternehmens entsprechen. 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 so potenzielle, im Produktionssystem auftretende Fehlerquellen erkannt werden. Darüber hinaus entfällt durch das automatische Abwickeln dieser Prozesse ein manuelles Aufsetzen der Testinfrastruktur, wobei CEs ihre IT-Ressourcen verstärkt auf Entwicklung neuer Services fokussieren können. Damit Unternehmen schnell von diesen Diensten profitieren können, ist es von essenzieller Bedeutung, dass nicht nur das Testen,

sondern ebenfalls die Bereitstellung neuer IT-Services kontinuierlich und automatisiert abgewickelt wird. Durch das unmittelbare Kompilieren, Validieren und Versionieren bei der Integration neuen Codes, steht zu jedem Zeitpunkt eine für die Veröffentlichung geeignete Anwendungsversion im zentralen Repository bereit. Unter Zuhilfenahme eines effektiven CD-Prozesses kann somit ein mehrfaches tägliches Ausrollen der Dienste ermöglicht werden. Dies führt zur Verkürzung des *Time-To-Values*, also dem Zeitintervall, bis der entwickelte IT-Service den ersten Kundennutzen herbeiführt [61]. Anstelle der Bereitstellung einer vollständig implementierten Anwendung bezweckt das CI/CD-Verfahren ein inkrementelles und frühes Ausrollen kleiner Features. Zwar besteht hierbei das Risiko, dass der Kundennutzen im Vergleich zur Komplett Einführung zunächst abgeschwächt ist, jedoch ermöglicht diese Früheinführung eine zügige Akkumulation und Umsetzung des Anwenderfeedbacks [14, S. 9]. Entschließt sich ein CE dazu, Bereitstellungsprozesse zu automatisieren, sollte dieses strategisch 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 [99, 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 dabei 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 eine 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 das 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 Programmbibliothek gewährleistet, dass sämtliche Compliance-Standards der SAP, wie Sicherheits- und Datenschutzbestimmungen, Branchenvorschriften oder interne Richtlinien bei der Entwicklung

von IT-Services 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 tätig, bei welchen ebenfalls strikte Regularien gelten. Durch die Bereitstellung dieser standardisierten Bibliothek entfällt für Unternehmensentwickler somit die Notwendigkeit einer zeitaufwendigen Implementierung dieser Schritte. Darüber hinaus werden für Azure Pipelines umfassende Bereitstellungsfunktionalitäten zur Verfügung gestellt. Dazu gehört das Blue/Green-Deployment. Dieses stellt eine effektive Möglichkeit dar, Software unter Produktionsbedingungen zu testen und kritische Ausfallzeiten bei der Bereitstellung von Software zu vermeiden. Dies spielt insbesondere eine wichtige Rolle, wenn Fehler in einem Service eines ERP-Systems auftreten, jedoch das herkömmliche Bereitstellen einer neuen Anwendungsversion aufgrund der durch die Initialisierung bedingten Ausfallzeit, zu unflexibel ist. Ein weiteres mit Azure Pipelines kompatibles Bereitstellungskonzept ist das Multi-Cloud-Deployment. Mit diesem können einzelne Module 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 der eigenen Cloud-Plattform. Dieser Integrationsvorteil erstreckt sich ebenfalls auf andere Services, welche innerhalb des Azure-Ökosystems bereitgestellt werden. Dazu gehören Produkte wie eine Entwicklungsumgebung, ein Projektmanagement- und Monitoring-Tool sowie ein Artefakt-Repository. Durch die Verwendung dieser Integrationsmöglichkeiten kann eine Rationalisierung des Entwicklungsprozesses erreicht werden. So ist es Entwicklern etwa möglich, Tests der Integration-Pipeline unmittelbar aus der Visual-Studio-Code-Umgebung auszuführen oder den Build-Status verschiedener Pipelines in einem zentralen Monitoring-Tool zu überwachen. Eine weitere Rationalisierung des Bereitstellungsprozesses kann durch die Verwendung verschiedener Ausführungsmechanismen erzielt werden. 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 ein-

sparen. Dies erwies sich ebenfalls im One-Strike-Programm als einen bedeutenden Aspekt. In dem One-Strike-Programm 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. In diesem Zusammenhang waren ebenfalls Kostenüberlegungen ein ausschlaggebender Aspekt für den Wechsel auf Azure. Durch die Nutzung einer SaaS-basierten CI/CD-Lösung können erhebliche Einsparungen bei Aufwandspositionen wie 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 CEs 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 oder die Verwendung von Caching. Laut Experte 4 haben diese Konzepte dazu beitragen, dass der CI/CD-Prozess der SAP-Standardentwicklung um 35 Prozent beschleunigt wurde [93, Z. 58 ff.]. Obwohl Azure Pipelines die für eine CEA essenziellen Funktionalitäten bereitstellt, besteht die Möglichkeit, dass Unternehmen aufgrund situativer Gegebenheiten, stattdessen Jenkins oder SAP CI/CD verwenden sollten. Dafür ausschlaggebende Gründe werden in dem folgenden Entscheidungsbaum aufgeführt:

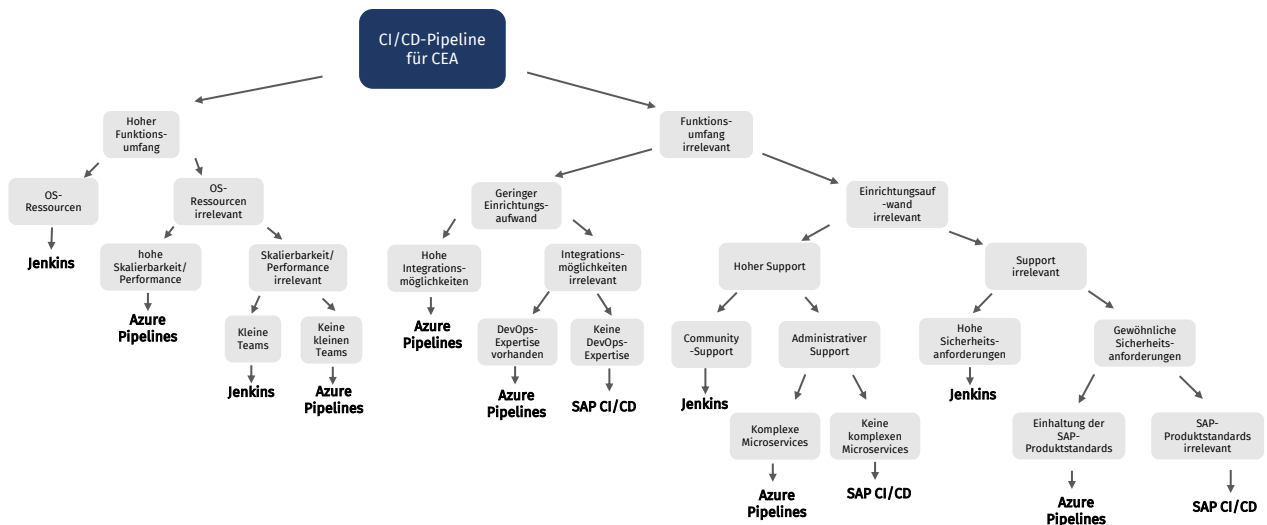


Abbildung 16: Entscheidungsbaum für die Wahl eines CI/CD-Pipeline-Tools.
Eigene Darstellung.

Laut Experte 1 sollte SAP CI/CD insbesondere von Kunden verwendet werden, welche über eine geringe Anzahl an Microservices verfügen und somit keine komplexen Anforderungen an den Bereitstellungsprozess besitzen [98, 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 [98, Z. 58 ff.]. Hierbei besitzen Entwicklungsteams 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. 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 Systemarchitektur ausrichten. Dies ermöglicht eine flexible Anpassung der Infrastrukturkomponenten, 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. Ein zusätzliches Argument für den Einsatz von Jenkins ist die Unabhängigkeit von den im Standard bereitgestellten Funktionalitäten. 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 Internetforen veröffentlichte Informationen. Dies unterstützt Entwickler dabei, Expertise, welche zur Implementierung einer maßgeschneiderten Pipeline benötigt wird, zu erlangen. Experte 4 bemerkt, dass Jenkins jedoch ausschließlich von kleinen Entwicklungsteams, welche mit einer geringen Anzahl an Technologien arbeiten, verwendet werden sollte [93, 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 hinauszuzögern. Dies kann dazu führen, dass potenzielle Sicherheitslücken und Fehler des Jenkins-Systems nicht behoben werden können. 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 und 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 Integration und Bereitstellung von IT-Services. Damit werden Code-Änderungen 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 eine CEA 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 eine CEA den größten Mehrwert birgt. Zur systematischen Evaluierung der Pipeline-Tools wurde ein Entscheidungs-Framework auf Grundlage des AHP-Verfahrens entworfen. In diesem Kontext erweist sich Azure Pipelines als das optimale CI/CD-Tool. Dies ist insbesondere auf den hohen Funktionsumfang des Systems zurückzuführen. So unterstützt Azure Pipelines die von der SAP veröffentlichte Programmbibliothek Project Piper, mit welcher für SAP-Technologien benötigte vorimplementierte Pipeline-Schritte ausgeliefert werden. Dies gewährleistet eine ressourcenoptimierte und stan-

dardisierte Implementierung der Bereitstellungs-Workflows. Sofern spezifische Funktionalitäten nicht durch Project Piper bereitgestellt werden, unterstützt der Azure-Pipelines-Standard darüber hinaus eine Vielzahl von Programmiersprachen, Plattformen und Test-Frameworks, welche eine maßgeschneiderte Implementierung von Pipelines ermöglichen. Zusätzlich stellt die hohe Flexibilität des CI/CD-Tools einen erheblichen Mehrwert dar. 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 Rechenkapazitäten während Stoßzeiten somit schnell und kosteneffektiv angepasst werden. Auch in Bezug auf die Performance erweist sich Azure Pipelines im Vergleich zu anderen CI/CD-Tools als besonders leistungsstark. 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. 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, erfordert das CI/CD-Tool bei der Pipeline-Implementierung eine deutlich geringere Expertise. 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 Me-

thode 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 [93, Z. 59 ff.].

6.2 Ausblick auf zukünftige Entwicklungen

Obwohl sich bereits eine Vielzahl verschiedener Bereitstellungs-Tools auf dem Markt etabliert haben, besteht eine signifikante Wahrscheinlichkeit, dass CI/CD in naher Zukunft einer disruptiven Veränderung unterzogen wird. Eines dieser innovativen Bereitstellungskonzepte vermuten Forschungsinstitute wie die Linux Foundation in der KI [52]. Valerie Silverthorne, Executive Editor bei GitLab, merkt an, dass diese Technologie das Potenzial besitzt, den Entwicklungsprozess zu revolutionieren und die Effizienz der Entwicklerteams zu erhöhen [75]. In diesem Kontext ergeben sich insbesondere zwei Anwendungsfelder. Dazu gehören eine *Intelligente Fehlererkennung* sowie eine *Generative Testerstellung*. Mit Hilfe der *Intelligenten Fehlererken-*

nung 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 [45]. Ein weiteres KI-Anwendungsgebiet liegt in der *Generativen Testerstellung*. Durch den Einsatz intelligenter Algorithmen besteht die Möglichkeit Testfälle zu generieren, um Schwachstellen im Code aufzudecken [50]. Diese umfassen neben Unit-, Integration- sowie E2E-Tests ebenfalls intelligente Security-Analysen. Damit kann 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. Neben diesen generativen Technologien vermuten DevOps-Spezialisten ebenfalls in blockchainbasierten CI/CD-Konzepten ein hohes Potenzial. Ein mögliches Anwendungsgebiet könnte die Versionierung von IT-Services darstellen. Dabei werden Metadaten von Code-Änderungen durch eine kryptografische Signatur gesichert und unveränderlich auf der Blockchain abgelegt. Dies erhöht die Transparenz über den gesamten Anwendungsverlauf hinweg, womit die Probleme effizienter identifiziert und Rollbacks auf frühere Versionen leichter durchgeführt werden können. Darüber hinaus kann diese unveränderlich Protokollierung ebenfalls im Rahmen der CI/CD-Validierungsprozesse eingesetzt werden. So lassen sich Testergebnisse sowie während des Code-Reviews hervorgegangene Kommentare, Bewertungen und Genehmigungen als Transaktion in der Blockchain ablegen. Damit ist es möglich, Manipulationen und Fälschungen während des Bereitstellungsprozesses zu verhindern und somit die Sicherheit der IT-Services zu erhöhen. Obwohl zahlreiche DevOps-Experten ein großes Potenzial in KI- sowie Blockchain-gestützten CI/CD-Prozessen erkennen, bleibt offen, wie sich diese Konzepte in der praktischen Anwendung bewähren. Dabei sollte etwa evaluiert werden, wie sich diese Methoden 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?

Literaturverzeichnis

Print-Quellen

- [1] Aghamanoukjan, A., Buber, R. und Meyer, M. „Qualitative Interviews“. In: *Qualitative Marktforschung*. Hrsg. von Buber, R. und Holzmüller, H. Wiesbaden: Gabler Verlag / GWV Fachverlage GmbH, Wiesbaden, 2009, S. 415–436. ISBN: 978-3-8349-0976-3.
- [2] Attardi, J. „Introduction to Netlify CMS“. In: *Using Gatsby and Netlify CMS*. Hrsg. von Attardi, J. Berkeley, Kalifornien: Apress und Imprint: Apress, 2020, S. 1–12. ISBN: 978-1-4842-6296-2. DOI: 10.1007/978-1-4842-6297-9_1.
- [3] Balalaie, A., Heydarnoori, A. und Jamshidi, P. „Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture“. In: *IEEE Software* 33.3 (2016), S. 42–52. ISSN: 0740-7459. DOI: 10.1109/MS.2016.64.
- [4] Belmont, J.-M. *Hands-on continuous integration and delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI*. Birmingham, England: Packt Publishing, 2018. ISBN: 9781789133073.
- [5] Biehl, M. *API architecture: The big picture for building APIs*. API-university series. API-University Press, 2015. ISBN: 9781508676645.
- [6] Blischak, J. D., Davenport, E. R. und Wilson, G. „A Quick Introduction to Version Control with Git and GitHub“. In: *PLoS computational biology* (2016). DOI: 10.1371/journal.pcbi.1004668.
- [7] Bruns, R. und Dunkel, J. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Xpert.press. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 9783642024399. DOI: 10.1007/978-3-642-02439-9.
- [8] De Lauretis, L. „From Monolithic Architecture to Microservices Architecture“. In: *2019 IEEE International Symposium on Software Reliability En-*

- gineering Workshops (ISSREW)*. IEEE, 10/27/2019 - 10/30/2019, S. 93–96. ISBN: 978-1-7281-5138-0. DOI: 10.1109/ISSREW.2019.00050.
- [9] Ellram, L. „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.
 - [10] Engström, E., Runeson, P. und Skoglund, M. „A systematic review on regression test selection techniques“. In: *Information and Software Technology* 52.1 (2010), S. 14–30. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2009.07.001. URL: <https://www.sciencedirect.com/science/article/pii/S0950584909001219>.
 - [11] Gloger, B. „Scrum“. In: *Informatik-Spektrum* 33.2 (2010), S. 195–200. ISSN: 0170-6012. DOI: 10.1007/s00287-010-0426-6.
 - [12] Gloger, B. *Scrum: Produkte zuverlässig und schnell entwickeln*. 5. Auflage. Hanser eLibrary. München: Hanser, 2016. ISBN: 9783446448360. DOI: 10.3139/9783446448360.
 - [13] Goll, J. und Hommel, D. *Mit Scrum zum gewünschten System*. Wiesbaden: Springer Vieweg, 2015. ISBN: 9783658107208. DOI: 10.1007/978-3-658-10721-5.
 - [14] Halstenberg, J. *DevOps: Ein Überblick*. Essentials Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020. ISBN: 9783658314057. URL: <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6380828>.
 - [15] Hüttermann, M. *DevOps for Developers*. The expert’s voice in Web development DevOps for developers. Berkeley, Kalifornien: Apress und Imprint: Apress, 2012. ISBN: 9781430245704.
 - [16] Ishizaka, A. und Labib, A. „Review of the main developments in the analytic hierarchy process“. In: *Expert Systems with Applications* (2011). ISSN: 09574174. DOI: 10.1016/j.eswa.2011.04.143.

- [17] Jamil, M. A. u. a. „Software Testing Techniques: A Literature Review“. In: *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*. IEEE, 11/22/2016 - 11/24/2016, S. 177–182. ISBN: 978-1-5090-4521-1. DOI: 10.1109/ICT4M.2016.045.
- [18] Kaiser, R. *Qualitative Experteninterviews: Konzeptionelle Grundlagen und praktische Durchführung*. 2. Auflage. Elemente der Politik. Wiesbaden: Springer Fachmedien Wiesbaden und Imprint: Springer VS, 2021. ISBN: 9783658302559. DOI: 10.1007/978-3-658-30255-9.
- [19] Kratzke, N. und Quint, P.-C. „Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study“. In: *Journal of Systems and Software* 126 (2017), S. 1–16. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.01.001. URL: <https://www.sciencedirect.com/science/article/pii/S0164121217300018>.
- [20] Lange, M. u. a. „Modern Build Automation for an Insurance Company Tool Selection“. In: *Procedia Computer Science* 219 (2023), S. 736–743. ISSN: 1877-0509. DOI: 10.1016/j.procs.2023.01.346. URL: <https://www.sciencedirect.com/science/article/pii/S1877050923003551>.
- [21] Maximini, D. *Scrum-Einführung in der Unternehmenspraxis*. 2. Auflage. Berlin, Heidelberg: Springer Gabler, 2013. ISBN: 978-3-662-56325-0. URL: <https://link.springer.com/content/pdf/10.1007/978-3-662-56326-7.pdf>.
- [22] Rangnau, T. u. a. „Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines“. In: *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 10/5/2020 - 10/8/2020, S. 145–154. ISBN: 978-1-7281-6473-1. DOI: 10.1109/EDOC49727.2020.00026.
- [23] Reinheimer, S. *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.

- [24] Rudrabhatla, C. K. „Comparison of zero downtime based deployment techniques in public cloud infrastructure“. In: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. IEEE, 10/7/2020 - 10/9/2020, S. 1082–1086. ISBN: 978-1-7281-5464-0. DOI: 10.1109/I-SMAC49090.2020.9243605.
- [25] Saaty, T. L. „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.
- [26] Schmiedmayer, P. u. a. „Global Software Engineering in a Global Classroom“. In: *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 5/22/2022 - 5/24/2022, S. 113–121. ISBN: 978-1-6654-9592-9. DOI: 10.1109/ICSE-SEET55299.2022.9794211.
- [27] Vesey, J. T. „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>.
- [28] Vieweg, W. „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.
- [29] Vivenzio, A., Hrsg. *Testmanagement Bei SAP-Projekten: Erfolgreich Planen, Steuern, Reporten Bei der Einführung Von SAP-Banking*. Wiesbaden: Springer Wiesbaden, 2013. ISBN: 978-3-8348-1623-8. DOI: 10.1007/978-3-8348-2142-3.
- [30] Zampetti, F. 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

- [31] abdalslam. *Continuous Integration (CI) Statistics, Trends And Facts 2023*. Abdalasal. 2023. URL: https://abdalslam.com/continuous-integration-ci-statistics?utm_content=cmp-true (besucht am 20.04.2023).
- [32] Adam, J. *Was ist agile Softwareentwicklung?* K&C. 2021. URL: <https://kruschecompany.com/de/agile-softwareentwicklung/> (besucht am 05.03.2023).
- [33] Arora, S. *What Is and What Are the Benefits of Docker Container?* Simplilearn. 2023. URL: <https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container> (besucht am 09.04.2023).
- [34] Atta, S. *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).
- [35] *Azure DevOps Services Pricing*. Microsoft. 2023. URL: <https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/> (besucht am 10.04.2023).
- [36] *Azure Pipelines Extensions*. Microsoft. 2023. URL: <https://marketplace.visualstudio.com/search?target=AzureDevOps&category=Azure%20Pipelines&sortBy=Installs> (besucht am 10.04.2023).
- [37] Bhandal, S. und Taylor, A. *The Evolution from Agile to DevOps to Continuous Delivery — Qentelli*. Qentelli. 2023. URL: <https://www.qentelli.com/thought-leadership/insights/evolution-agile-devops-continuous-delivery> (besucht am 05.03.2023).
- [38] *Bitbucket OAuth*. Jenkins. 2023. URL: <https://plugins.jenkins.io/bitbucket-oauth/> (besucht am 19.04.2023).
- [39] Bose, S. *What is End To End Testing?* BrowserStack. 2023. URL: <https://www.browserstack.com/guide/end-to-end-testing> (besucht am 08.03.2023).
- [40] Buchanan, I. *Feature Flags*. Atlassian. 2023. URL: <https://www.atlassian.com/continuous-delivery/principles/feature-flags> (besucht am 09.04.2023).

- [41] *Build Stage - Project Piper: Continuous Delivery for the SAP Ecosystem*. Project Piper. 2023. URL: <https://www.project-piper.io/stages/build/> (besucht am 10.04.2023).
- [42] Cecchini, A. *What It Is and Why It's Important - Part 2*. ITPFED. 2021. URL: <https://itpfed.com/composable-erp-what-it-is-and-why-its-important-part-2/> (besucht am 18.04.2023).
- [43] *CI/CD Pipelines Archives*. Codefresh. 2023. URL: <https://codefresh.io/learn/ci-cd-pipelines/> (besucht am 19.04.2023).
- [44] *cloudFoundryDeploy - Project Piper: Continuous Delivery for the SAP Ecosystem*. Project Piper. 2023. URL: <https://www.project-piper.io/steps/cloudFoundryDeploy/#additional-hints> (besucht am 10.04.2023).
- [45] *Code Defect AI from Microsoft AI Lab*. Microsoft. URL: <https://www.microsoft.com/en-us/ai/ai-lab-code-defect> (besucht am 19.04.2023).
- [46] 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).
- [47] Danielson, S. *Konfigurieren von Pipelinetriggern - Azure Pipelines*. Microsoft. 2023. URL: <https://learn.microsoft.com/de-de/azure/devops/pipelines/process/pipeline-triggers?view=azure-devops> (besucht am 10.04.2023).
- [48] Danielson, S. *Microsoft-hosted agents for Azure Pipelines - Azure Pipelines*. Microsoft. 2023. URL: <https://learn.microsoft.com/en-us/azure/devops/pipelines/licensing/concurrent-jobs?view=azure-devops&tabs=ms-hosted> (besucht am 10.04.2023).
- [49] Denning, S. *Beyond Agile Operations: How To Achieve The Holy Grail Of Strategic Agility*. 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).

- [50] Fernandes, H. *AI In Test Automation: Here's How It Works*. QALead. URL: <https://theqalead.com/automation-ai/ai-test-automation/> (besucht am 19.04.2023).
- [51] Fowler, M. *Continuous Integration*. MartinFowler. 2023. URL: <https://martinfowler.com/articles/continuousIntegration.html> (besucht am 19.04.2023).
- [52] *Future of Continuous Delivery Trends - CD Foundation*. The Linux Foundation. 2022. URL: <https://cd.foundation/blog/2022/04/01/future-of-continuous-delivery-trends/> (besucht am 16.04.2023).
- [53] Galer, S. *Disrupted ERP Systems Rise to Reckoning in the Cloud*. SAP. 2022. URL: <https://news.sap.com/2022/10/composable-business-for-disrupted-erp-systems/> (besucht am 08.04.2023).
- [54] Gaughan, D. u. a. *Future of Applications: Delivering the Composable Enterprise*. Gartner. 2020. URL: <https://www.gartner.com/en/doc/465932-future-of-applications-delivering-the-composable-enterprise> (besucht am 18.04.2023).
- [55] *GitHub Authentication*. Jenkins. 2023. URL: <https://plugins.jenkins.io/github-oauth/> (besucht am 10.04.2023).
- [56] Gosh, S. *Top 10 Jenkins Plugins with Features for 2023*. Knowledgehut. 2023. URL: <https://www.knowledgehut.com/blog/devops/jenkins-plugins> (besucht am 10.04.2023).
- [57] Hamilton, T. *Manual Testing Tutorial: What is, Types, Concepts*. Guru99. 2020. URL: <https://www.guru99.com/manual-testing.html> (besucht am 19.04.2023).
- [58] *IBM Documentation - Monitoring Platform*. IBM. 2023. URL: <https://www.ibm.com/docs/en/cloud-paks/cp-data/4.6.x?topic=2-monitoring-platform> (besucht am 19.04.2023).

- [59] *IT Spending and Staffing Benchmarks 2022/2023*. Computer Economics. 2023. URL: <https://avasant.com/report/it-spending-and-staffing-benchmarks-2022-2023-chapter-1-executive-summary/> (besucht am 19.04.2023).
- [60] Klingberg, J. *Composable Enterprise: Warum das Unternehmen der Zukunft modular aufgebaut ist*. Magnolia. 2021. URL: https://www.magnolia-cms.com/de_DE/blog/composable-enterprise.html (besucht am 13.03.2023).
- [61] Kreutzer, R. *Definition: Time-to-Value*. Gabler Wirtschaftslexikon. URL: <https://wirtschaftslexikon.gabler.de/definition/time-value-54272> (besucht am 19.04.2023).
- [62] *Multitarget Applications in the Cloud Foundry Environment*. SAP SE. 2023. URL: <https://help.sap.com/docs/btp/sap-business-technology-platform/multitarget-applications-in-cloud-foundry-environment> (besucht am 09.04.2023).
- [63] *Netflix Architecture: How Much Does Netflix's AWS Cost?* CloudZero. 2021. URL: <https://www.cloudzero.com/blog/netflix-aws> (besucht am 19.04.2023).
- [64] Panetta, K. *Top 10 Strategic Predictions for 2022 and Beyond*. Gartner. 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).
- [65] *Parameterized Trigger*. Jenkins. 2023. URL: <https://plugins.jenkins.io/parameterized-trigger/> (besucht am 19.04.2023).
- [66] Paspelava, D. *What is Unit Testing in Software*. Exposit. 2021. URL: <https://www.exposit.com/blog/what-unit-testing-software-testing-and-why-it-important/> (besucht am 08.03.2023).
- [67] *Posts containing 'azure pipelines' - Stack Overflow*. Stack Overflow. 2023. URL: <https://stackoverflow.com/search?q=azure+Pipelines> (besucht am 03.04.2023).

- [68] *Posts containing 'jenkins' - Stack Overflow*. Stack Overflow. 2023. URL: <https://stackoverflow.com/search?q=jenkins> (besucht am 03.04.2023).
- [69] *Posts containing 'sap ci/cd' - Stack Overflow*. Stack Overflow. 2023. URL: <https://stackoverflow.com/search?q=SAP+ci%2FCD> (besucht am 03.04.2023).
- [70] *Release Stage - Project Piper: Continuous Delivery for the SAP Ecosystem*. Project Piper. 2023. URL: <https://www.project-piper.io/stages/release/> (besucht am 10.04.2023).
- [71] Reynolds, J. *What Is a Staging Environment? How to Get It Right*. Plutora. 2022. URL: <https://www.plutora.com/blog/what-staging-environment-how-get-it-right> (besucht am 19.04.2023).
- [72] *Role-based Authorization Strategy*. Jenkins. 2023. URL: <https://plugins.jenkins.io/role-strategy/> (besucht am 10.04.2023).
- [73] *SAML Single Sign On(SSO)*. Jenkins. 2023. URL: <https://plugins.jenkins.io/miniorange-saml-sp/> (besucht am 19.04.2023).
- [74] Schönenstein, J. *Composable-Business und -Commerce durch MACH-Architektur*. communicode AG. 2023. URL: <https://www.communicode.de/blog/work/composable-business-und-commerce-durch-mach-technologie> (besucht am 13.03.2023).
- [75] Silverthorne, V. *Why AI in DevOps is here to stay*. GitLab. 2022. URL: <https://about.gitlab.com/blog/2022/09/15/why-ai-in-devops-is-here-to-stay/> (besucht am 16.04.2023).
- [76] *Stack Overflow Developer Survey 2020*. Stack Overflow. 2020. URL: <https://insights.stackoverflow.com/survey/2020> (besucht am 03.04.2023).
- [77] Stevens, H. *Interplay of SAP Cloud Platform Transport Management, CTS+ and ChaRM in hybrid landscapes*. SAP Blogs. 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).

- [78] *The Cloud vs. On-Premise Cost: Which One is Cheaper?* Executech. 2023. URL: <https://www.executech.com/insights/the-cloud-vs-on-premise-cost-comparison/> (besucht am 10.04.2023).
- [79] *The Evolution of Container Usage at Netflix - Netflix TechBlog*. Netflixtechblog. 2017. URL: <https://netflixtechblog.com/the-evolution-of-container-usage-at-netflix-3abfc096781b> (besucht am 10.04.2023).
- [80] *The Future is Composable*. Sensedia. 2020. URL: <https://f.hubspotusercontent30.net/hubfs/4209582/%5BInternational%5D%20Boardroom%20Nordics/%28EN%29%20Composable%20Enterprises%20-%20Sensedia.pdf> (besucht am 10.04.2023).
- [81] Ugochi, U. *Deployment Strategies: 6 Explained in Depth*. Plutora. 2022. URL: <https://www.plutora.com/blog/deployment-strategies-6-explained-in-depth> (besucht am 08.03.2023).
- [82] *Using Jenkins agents*. Jenkins. 2023-04-10. URL: <https://www.jenkins.io/doc/book/using/using-agents/> (besucht am 10.04.2023).
- [83] *Vergleich der Azure-Supportpläne*. Azure. 2023. URL: <https://azure.microsoft.com/de-de/support/plans> (besucht am 10.04.2023).
- [84] *Was ist die Definition of Done (DoD)?* Scrumevents. 2023. URL: <https://www.scrum-events.de/was-ist-die-definition-of-done-dod.html> (besucht am 19.04.2023).
- [85] *Was ist eine Multi-Cloud?* Red Hat. 2022. URL: <https://www.redhat.com/de/topics/cloud-computing/what-is-multicloud> (besucht am 19.04.2023).
- [86] *What is Application Monitoring? — VMware Glossary*. VMware. 2022. URL: <https://www.vmware.com/topics/glossary/content/application-monitoring.html> (besucht am 19.04.2023).
- [87] *What is Azure DevOps Pipeline*. Intellipaat. 2023. URL: <https://intellipaat.com/blog/azure-devops-pipelines/> (besucht am 19.04.2023).

- [88] *What Is CI/CD and How Does It Work?* Synopsys. 2023. URL: <https://www.synopsys.com/glossary/what-is-cicd.html> (besucht am 08.03.2023).
- [89] *What is Infrastructure Monitoring? How it Works and Use Cases*. Datadog. 2021. URL: <https://www.datadoghq.com/knowledge-center/infrastructure-monitoring/> (besucht am 19.04.2023).
- [90] *What Is SAP Continuous Integration and Delivery*. SAP SE. 2023. URL: https://help.sap.com/docs/CONTINUOUS_DELIVERY/99c72101f7ee40d0b2deb4df72ba1a618ca03fdca24e56924cc87cfbb7673a.html?language=en-US (besucht am 09.04.2023).

Literaturverzeichnis

- [91] Backend-Test-Entwickler SAP DTS Integration - Experte 7. *Expertengewichtung 4*. 29.03.2023.
- [92] Frontend-Test-Entwickler SAP Hyperspace Adoption & Onboarding - Experte 4. *Expertengewichtung 3*. 22.03.2023.
- [93] Frontend-Test-Entwickler SAP Hyperspace Adoption & Onboarding - Experte 4. *Experteninterview 4*. 22.03.2023.
- [94] Full-Stack-Entwickler SAP DTS Integration - Experte 6. *Expertengewichtung 2*. 21.03.2023.
- [95] Product Manager SAP CLM - Experte 8. *Expertengewichtung 5*. 29.03.2023.
- [96] Product Manager SAP Hyperspace CI/CD - Experte 2. *Experteninterview 2*. 24.03.2023.
- [97] Product Manager SAP Hyperspace Security Tools - Experte 3. *Experteninterview 3*. 22.03.2023.
- [98] Product Owner SAP BTP Prod and Infra - Experte 1. *Experteninterview 1*. 17.03.2023.

- [99] Software Architekt SAP DTS Integration - Experte 5. *Expertengewichtung 1*.
27.03.2023.

Anhang

Anhangsverzeichnis

A	Allgemeine Ergänzungen	XXIV
A.1	DevOps	XXIV
A.2	Ramped-Deployment	XXV
A.3	Feature Flags	XXVI
B	Pipeline-Prototyp	XXVII
B.1	Azure Pipelines	XXVII
B.2	Jenkins	XXXIV
B.3	SAP CI/CD	XXXVII
C	Expertenmaterialien	XLII
C.1	Experteninterview 1	XLII
C.2	Experteninterview 2	XLVII
C.3	Experteninterview 3	L
C.4	Experteninterview 4	LII
C.5	Kodierung der Experteninterviews	LVI
C.6	Expertengewichtung 1	LXVIII
C.7	Expertengewichtung 2	LXXIII
C.8	Expertengewichtung 3	LXXVIII
C.9	Expertengewichtung 4	LXXXIII
C.10	Expertengewichtung 5	LXXXVIII
C.11	Expertengewichtung Durchschnitt	XCIII

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)* [14, 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 [14, 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 [14, S. 7]. Der *Time-to-Market* beschreibt die Zeitspanne zwischen Entwicklungsentstehungsprozess und der Markteinführung von IT-Services [27, 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 [14, 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 [27]. Aus diesen Erfahrungen sollen Best-Practises entwickelt werden, welche innerhalb von Teams und organisationsübergreifend weitergegeben werden (*Teilen*) [14, S. 7].

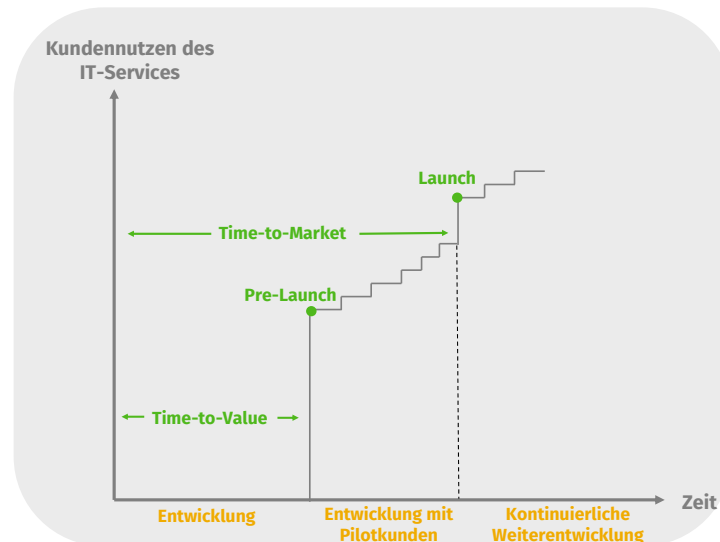


Abbildung 17: Zeitliche Darstellung der Herbeiführung von Kundennutzen bei der Entwicklung von IT-Services [14, S. 9].

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 [81].

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 [40].

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 18: 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 19: 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 20: 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 21: 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 22: 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 23: Integration- und Delivery-Zeit für den PlantService mit Jenkins. Eigene Darstellung.

B.3 SAP CI/CD


B.3.1 CI-Pipeline-Konfiguration



Abbildung 24: 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

OFF

Lint Check

ON

☐ Fail on Error

Additional Unit Tests

npm Script: mykarma

ON

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

ON

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

ON

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:

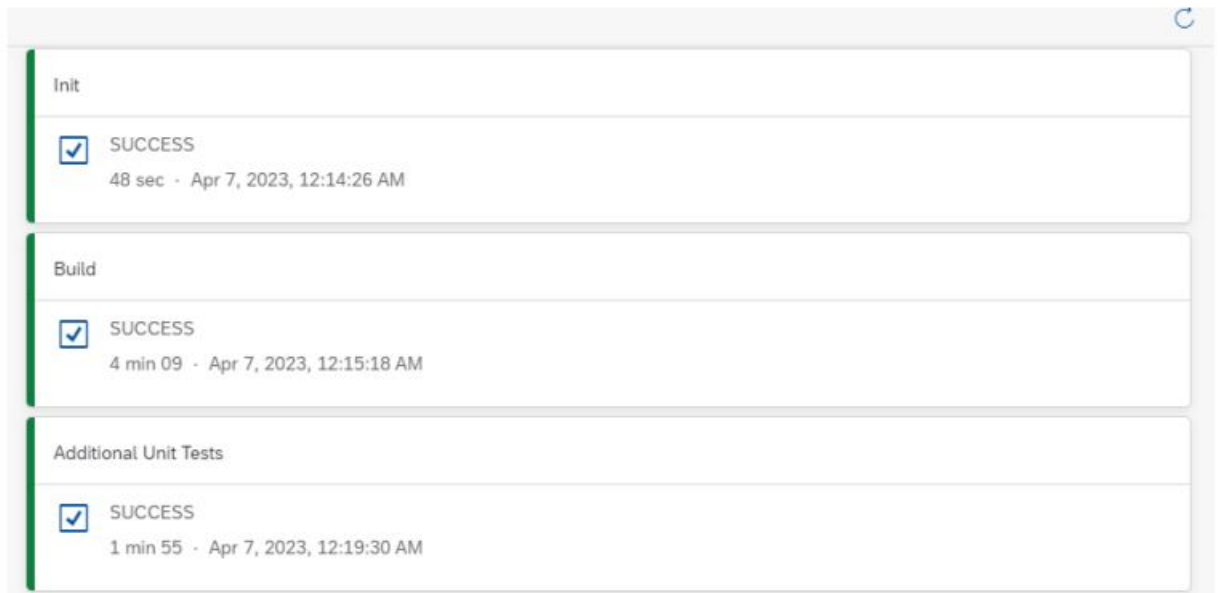
OFF

Service Key:

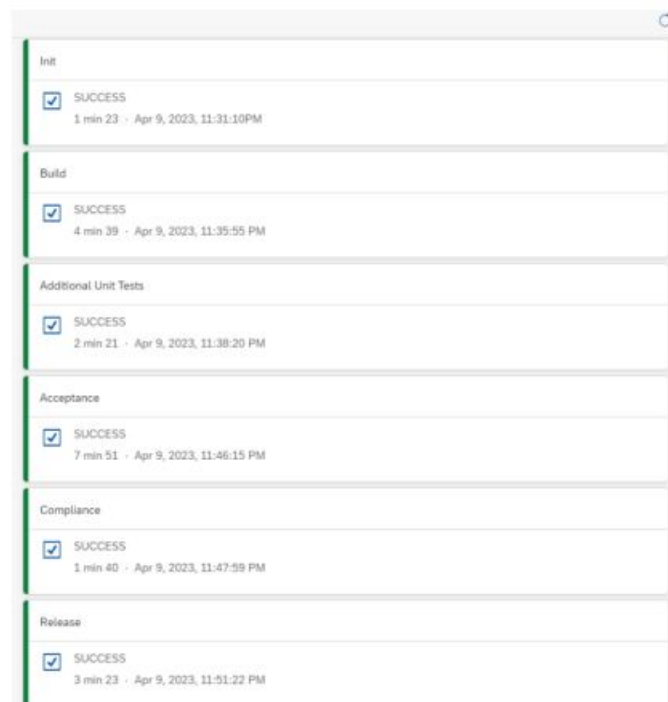
Abbildung 25: CD-Pipeline-Konfiguration für SAP CI/CD

XXXVIII

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 26: 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 27: 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 28: 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	2	1	2	2	2	0,1852
Integrationsmöglichkeiten	1	1	2	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	1	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	1	0	0	0	0,1111
Flexibilität	1	1	2	1	1	0	1	1	0,1235
Support	1	2	2	2	2	1	1	1	0,1728
Sicherheit	1	1	2	1	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 Manager SAP 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	1	0	2	0,1728
Integrationsmöglichkeiten	0	1	2	2	2	2	2	2	0,1852
Kosten	0	0	1	2	1	2	2	2	0,1235
Skalierbarkeit	0	0	0	1	2	2	2	2	0,1358
Performance	0	0	0	2	0	2	2	2	0,0741
Flexibilität	0	0	0	2	1	2	1	2	0,1358
Support	1	0	1	2	0	1	1	2	0,0617
Sicherheit	2	0	0	0	0	1	1	2	0,0988
Benutzerfreundlichkeit	0	0	0	0	1	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

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
Communitiy-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

