WHITEPAPER

DevOps und SAP HANA SQL Data Warehousing

von Martin Peitz | ISR Information Products AG

INHALT

Einleitung ... 3

1 | DevOps Philosophie ... 4

1.1 | Prinzipien und Werte ... 4

1.2 | Phasen, Prozesse und Tools ... 6

1.2.1 | Continuous Integration ... 6

1.2.2 | Continuous Delivery ... 7

1.2.3 | Continuous Testing ... 8

1.2.4 | Continuous Feedback ... 8

1.2.5 | Tools ... 9

1.3 | Business Investition in DevOps ... 9

1.4 | Business Mehrwert durch DevOps ... 10

1.5 | DevOps auf einen Blick ... 12

2 | DevOps im Data-Warehouse-Kontext ... 13

2.1 | Anforderungen an das zukunftsfähige Data Warehouse ... 13

2.2 | Klassische vs. DevOps-DWH-Entwicklung ... 15

2.3 | Verteilte Source-Code-Verwaltung ... 17

2.4 | SAP SQL Data Warehousing ... 19

2.4.1 | Continuous Integration ... 20

2.4.2 | Continuous Delivery ... 21

2.4.3 | Continuous Testing ... 22

2.4.4 | Continuous Feedback ... 23

2.4.5 | DevOps-Phasen ... 24

2.4.6 | Tools ... 26

2.4.7 | Investition SAP SQL Data Warehousing ... 26

2.4.8 | Vorteile des SAP SQL Data Warehousing ... 27

Fazit ... 28

Über ISR ... 29

EINLEITUNG

DevOps gehört in den vergangenen Jahren zu einem der beliebtesten Schlagwörter der Softwareentwicklung. Ein fundiertes Verständnis scheint allerdings noch nicht weit verbreitet, wie ein Blick auf die Google Trends zum Thema DevOps zeigt (Abb. 1.1). Dies gilt insbesondere im Data-Warehouse-Umfeld, in dem traditionelle Entwicklungsmethoden weiterhin dominieren und das Potential von DevOps erst langsam erschlossen wird.

In diesem Whitepaper möchten wir DevOps erläutern und Ihnen die grundlegenden Ziele und Charakteristika sowie die Herausforderungen und Vorteile dieses Entwicklungsansatzes näherbringen.

Nach der allgemeinen Einführung in die DevOps-Philosophie, widmen wir uns den gegenwärtigen und absehbaren Einflussfaktoren des Themengebiets Data Warehouse und beschreiben den Einfluss, den DevOps neuerdings auch auf die Entwicklung des Data Warehouse nimmt. Dazu stellen wir den klassischen Data-Warehouse-Entwicklungsansatzes der Softwareorientierten DevOps-Data-Warehouse-Entwicklung gegenüber und gehen im Anschluss auf die entsprechende Lösung aus dem Hause SAP, das SAP SQL Data Warehousing ein.

Dabei wird deutlich, dass das SAP SQL Data Warehousing durch seine modulare und offene Struktur ideale Vorrausetzungen zur Anwendung der DevOps-Philosophie bietet und sich durch das Zusammenspiel von DevOps und der SAP-HANA-Technologie viele Vorteile für die absehbaren Herausforderungen im Bereich des Data Warehousing ergeben.

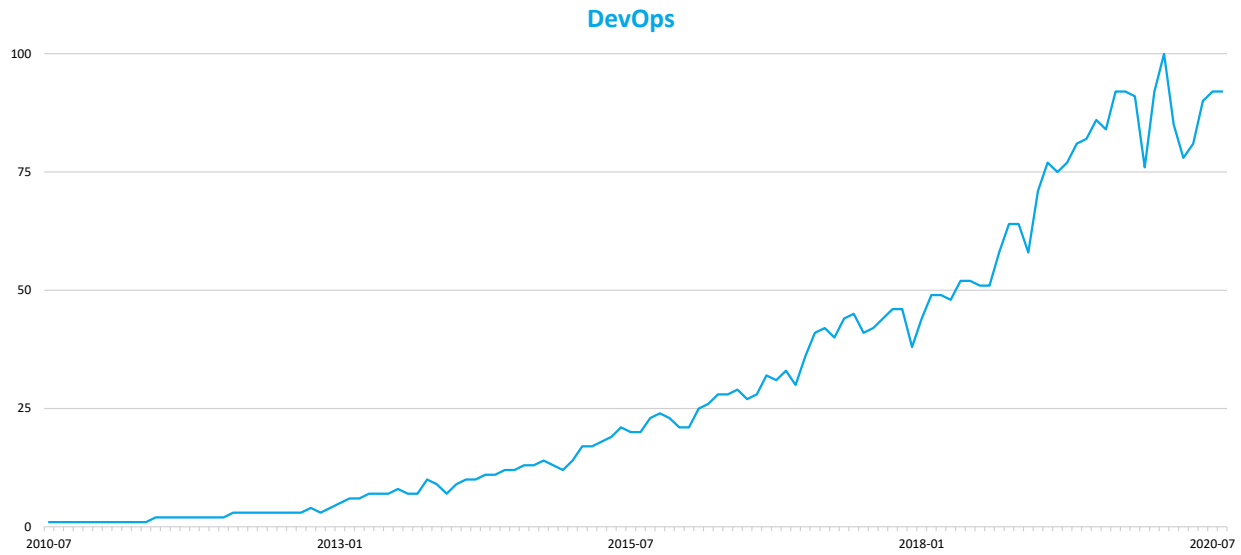

Abb. 1: Google Trends weltweit zu DevOps (https://trends.google.de/trends/explore?date=all&q=%2Fm%2F0c3tq11)

1 | DEVOPS-PHILOSOPHIE

DevOps beruht vor allem auf Prinzipien des **Lean Managements** und des **Agile Manifesto** und verfolgt im Kern das Ziel, unternehmerischen Erfolg in der Softwareentwicklung durch die kontinuierliche und effiziente Erbringung von Mehrwert oder Nutzen für den Kunden bzw. Anwender zu gewährleisten. In diesem Zusammenhang ist es bedeutend DevOps als **Philosophie oder Kultur** zu begreifen und es nicht auf die Verwendung bestimmter Methoden oder Tools zu reduzieren.

Zur Verdeutlichung dieses Gesamtkontextes stellen wir Ihnen daher zunächst die grundlegenden DevOps-Prinzipien vor. Im Anschluss werden spezifische Prozesse, Methoden und Tools erläutert.

1.1 | PRINZIPIEN UND WERTE

Wie der Name DevOps bereits nahelegt, geht es zentral darum, die Bereiche Entwicklung (Development) und Betrieb (Operations) besser zu integrieren und

- durch das Prinzip des **Arbeitsflusses (Flow)**, die Auslieferung von Software von der Entwicklung über den Betrieb bis zum Kunden/Anwender zu beschleunigen (Weg 1, Abb. 1.1)
- durch das Prinzip des **Feedbacks**, speziell zwischen den Bereichen Development und Operations, zuverlässigere Arbeitssysteme zu schaffen (Weg 2, Abb. 1.1)
- durch das Prinzip des **kontinuierlichen Lernens und Experimentierens,** eine Kultur des Vertrauens zu etablieren, so dass das Übernehmen von Risiken Teil der täglichen Arbeit wird (Weg 3, Abb. 2.1).

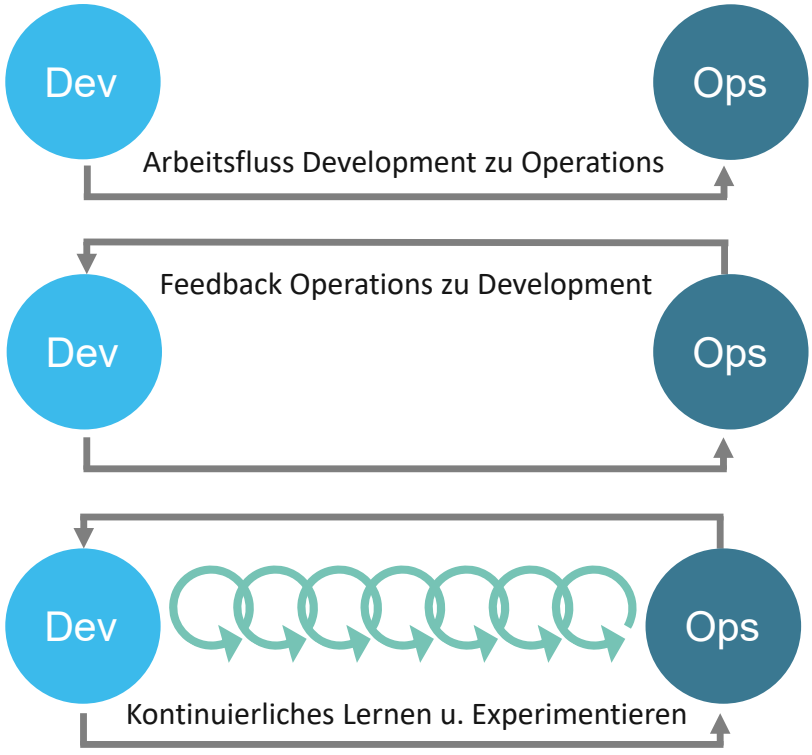

Abb. 1.1: Die drei DevOps-Wege

Der erste Schritt ist eine ganzheitliche Betrachtung des Lebenszyklus einer Softwareentwicklung. Aus Kunden- bzw. Anwendersicht endet eine Entwicklung mit der Fertigstellung in produktiven Umgebungen und sieht keine Brüche zwischen Testumgebungen in der Entwicklung und der finalen Administration im Bereich Operations vor. Die Vermeidung dieser Brüche und die **Überwindung der typischen Silo-Konstellationen** in der Organisation von Development und Operations ist daher ein wesentlicher Schritt, um organisatorische Hürden zu eliminieren, die Durchlaufzeiten zu verkürzen und so schneller relevante Mehrwerte für die Nutzer zu erzeugen und bereitzustellen.

Ein wichtiges Element zur Unterstützung dieser Beziehung ist schnelles und kontinuierliches Feedback. Wie fast alle modernen Philosophien zur Prozessverbesserung, setzt auch DevOps eine **Kultur des Diskurses und der Rückmeldung** voraus, um Fehler schnell zu erkennen. Dadurch kann ein wiederholtes Auftreten von Problemen frühzeitig vermieden werden.

Das Ziel ist die Schaffung sicherer Arbeitssysteme, die Probleme und Fehler in einem Stadium aufdecken, in dem sie keine allzu großen Auswirkungen auf die Durchlaufzeit haben, da sie schnell behoben werden können. Das dritte fundamentale Prinzip führt die zuvor genannten Aspekte konsequent fort und strebt eine **Kultur des Lernens und Vertrauens** an, in der die Beteiligten bereit sind Risiken einzugehen und Fehler zu begehen. Diese Entwicklung wird getragen von der Fokussierung auf kleine Schritte und Arbeitspakete, die Risiken minimieren.

Durch konsequente und dauerhafte Wiederholung von Durchlaufzyklen wirkt sich dies zudem positiv auf die gesamte Durchlaufzeit eines Projektes aus und schafft eine größere Flexibilität für laufende Änderungen und Anpassungen an neue Gegebenheiten.

1.2 | PHASEN, PROZESSE UND TOOLS

Die DevOps-Philosophie ist vor allem durch das Herunterbrechen der Entwicklungsaufgaben in kleine, voll auslieferbare Entwicklungsartefakte geprägt. Ziel ist die Veränderung des Entwicklungsansatzes von großen monolithischen Strukturen, mit wenigen, umfangreichen Durchlaufzyklen, hinzu kleinen agilen Servicearchitekturen, mit vielen Zyklen pro Tag. Eine eingängige Grafik (Abb. 1.2), die sich zu DevOps häufig findet, zeigt eine unendliche Schleife. Sie bildet die Verbindung der Bereiche Development und Operations mit den entsprechenden Phasen eines Durchlaufzyklus ab. Der entscheidende Faktor dieser andauernden Aufgabe ist **Kontinuität.**

Aus diesem Grund haben sich spezifische Kontinuitätsprozesse innerhalb von DevOps etabliert (Abb. 1.2), die die Phasen flankieren und insbesondere durch einen **hohen Automatisierungsgrad** den Prinzipien und Zielen von DevOps Rechnung tragen sollen.

1.2.1 | CONTINUOUS INTEGRATION

Der erste dieser Automatisierungsprozesse ist die kontinuierliche Integration oder **Continuous Integration**, kurz CI. Sie erfolgt im Bereich der Entwicklung und verlangt, dass der heute häufig von mehreren Entwicklern oder Entwicklerteams parallel erarbeitete Quellcode laufend zusammengefügt wird.

Der CI-Prozess sorgt dafür, dass alle Veränderungen des Quellcodes **automatisch** in einem gemeinsamen Repository münden und durch standardisierte Tests sichergestellt wird, dass keine Konflikte zwischen vorhandenem und neuem Code bestehen. Dadurch wird die Integrität der Entwicklungen laufend überprüft und gewährleistet, dass sich sämtliche Codebestandteile in das Gesamtgefüge des Produktes korrekt einfügen.

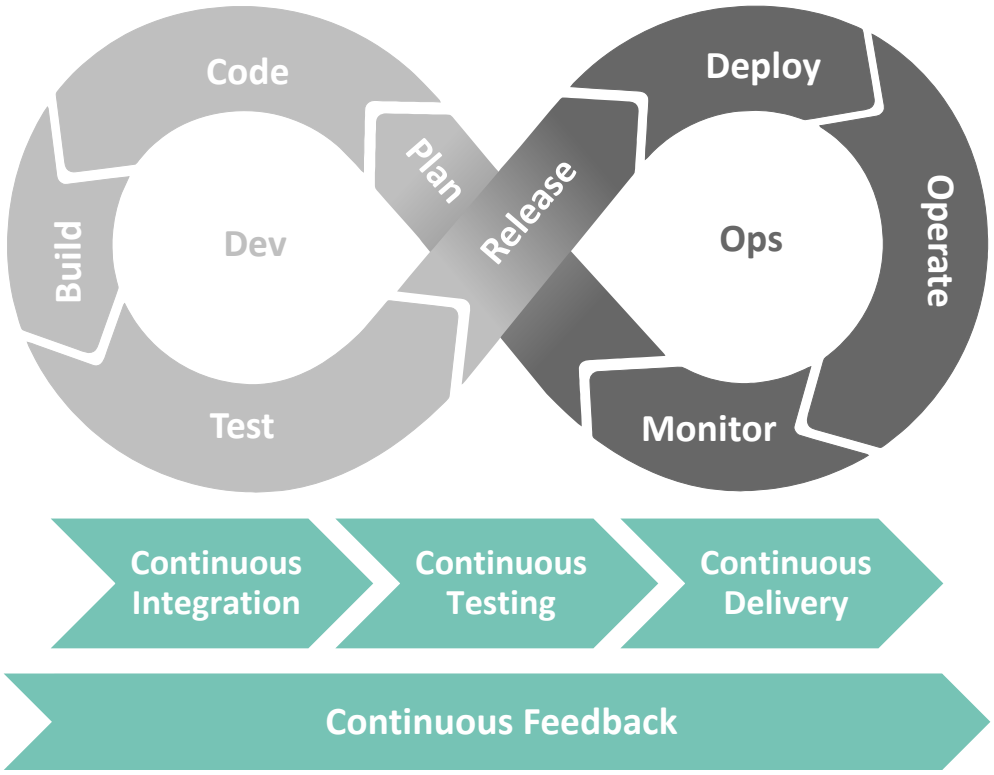

Abb. 1.2: DevOps-Phasen und Kontinuitätsprozesse

1.2.2 | CONTINUOUS DELIVERY

Der zweite Kontinuitätsprozess im Rahmen von DevOps ist Continuous Delivery, kurz CD. Dieser hat den Anspruch, dass Änderungen des Quellcodes bzw. neue Entwicklungen schnell, automatisiert und zuverlässig ausgeliefert werden. Dementsprechend geht es bei der Anwendung von Continuous Delivery um die Automatisierung der Auslieferungsprozesse in Entwicklungs-, Test- und Produktivumgebungen. Im Gesamtkontext von DevOps und der konsequenten Fortführung von Continuous Integration ist das Ziel, bereits kleine Entwicklungsartefakte zum frühestmöglichen Zeitpunkt in einen produktiven Zustand zu versetzen. Im Hinblick auf die finale Auslieferung in Produktivumgebungen werden dabei zwei Aspekte unterschieden (Abb. 1.3):

Ist die komplette Auslieferungskette automatisiert, wird von Continuous Deployment gesprochen. Continuous Delivery setzt den Schwerpunkt auf die Versetzung jedes Entwicklungsartefaktes in ein geprüft auslieferbares Stadium. Zur Risikominimierung wird der Produktivumgebung häufig noch eine identische Vorinstanz vorgeschoben.

Continuous Delivery umfasst nur die automatische Auslieferung bis auf diese Ebene. Das finale, operative Deployment erfolgt manuell, gegebenenfalls in zusammengefassten Paketen (Abb. 1.3). Hierfür kann es gute Gründe geben, wobei der Trend in der Softwareentwicklung vermehrt zum Continuous Deployment, als erweiterte Form des Continuous Delivery geht. Durch den grundlegenden Fokus auf die Auslieferungsautomatisierung wird jedoch primär Continuous Delivery als zweite wesentliche DevOps-Säule neben Continuous Integration genannt. Im Zusammenhang wird gerne von CI/CD gesprochen.

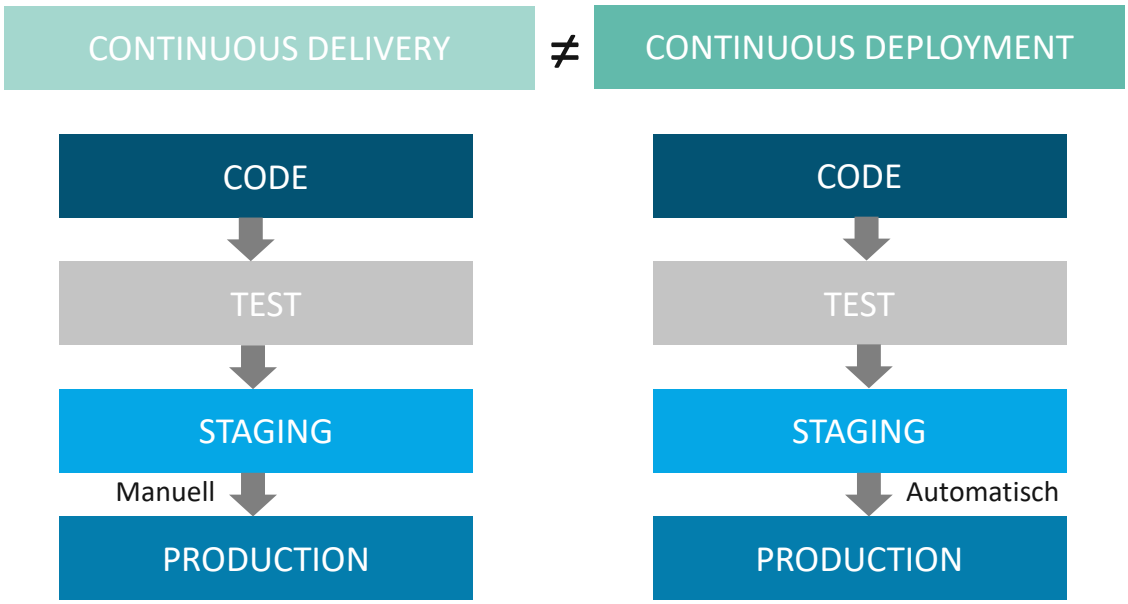

Abb. 1.3: Unterschied Continuous Delivery u. Continuous Deployment

1.2.3 | CONTINUOUS TESTING

Ein weiterer Prozess, der eine gewichtige Rolle für die erfolgreiche Anwendung von DevOps spielt, aber häufig übergangen wird, ist das kontinuierliche Testen oder Continuous Testing, kurz CT. Dies liegt daran, dass spezifische Tests bereits dem CI/CD Prozess zugeschrieben werden und Continuous Testing nur als beiläufiges Schlagwort neben der Testautomatisierung genannt wird. Dies verkennt allerdings den Schwerpunkt, den DevOps auf das proaktive Finden und Beheben von Fehlern legt, der über die Automatisierung von Prüfungen im Rahmen von CI/CD hinausgeht.

Continuous Testing kann als verbindendes Element von CI und CD betrachtet werden (Abb. 1.2) und zwar in dem Sinne, das Tests in DevOps nicht mehr nur eine Phase im Softwareentwicklungslebenszyklus darstellen, sondern eine **integrale Aktivität,** die sich in verschiedenen Überprüfungsformen und persönlichem Feedback in allen Phasen manifestiert. CT wird auf diese Weise zu einem **linearen Qualitätssicherungsprozess** innerhalb der DevOps-Zyklen.

1.2.4 | CONTINUOUS FEEDBACK

Der letzte Kontinuitätsaspekt ist als übergreifender Prozess elementarer Bestandteil aller DevOps-Phasen und spiegelt die Verbindung der Fachbereiche Development und Operations sowie der Nutzer bzw. Kunden wider.

Denn um die unendliche DevOps-Schleife mit Leben zu füllen, ist kontinuierliches Feedback unerlässlich. Sei es durch automatisierte Prüfungen oder durch Rückmeldungen beteiligter Stellen. Es bedarf der ständigen **Kommunikation,** um Anforderungen richtig zu verstehen und kontinuierilchen **Kontrolle,** ob die geplante Veränderung den erwogenen produktiven Mehrwert schafft. Hierdurch werden Fehlentwicklungen und Probleme frühestmöglich erkannt und kumulative Risiken minimiert.

1.2.5 | TOOLS

Unterschiedliche Tools unterstützen den DevOps-Prozess sehr stark, da sie sich weitestgehend für die **Automatisierung der Prozessschritte** verantwortlich zeigen. Die Tools finden zumeist in einem oder mehreren der skizzierten Prozessschritte Anwendung und sind entsprechend skalierbar. Darüber hinaus lassen sie sich durch offene Standardschnittstellen zumeist leicht miteinander verzahnen. Ziel ist es, manuelle, wiederkehrende Tätigkeiten entweder zu vereinfachen oder vollständig zu automatisieren. Die Liste verfügbarer Helfer des DevOps-Kosmos ist lang und es stehen sowohl kommerzielle als auch viele **Open-Source-Lösungen** bereit (Abb. 1.4). Dadurch kann die Tool-Auswahl sehr flexibel und bedürfnisgerecht erfolgen.

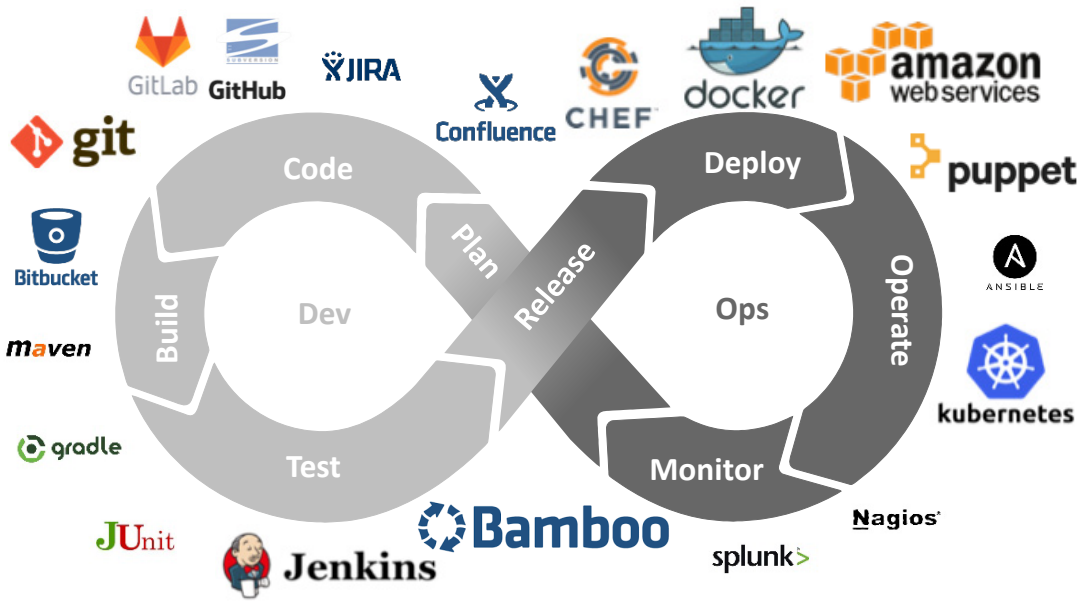

Abb. 1.4: Tools des DevOps-Kosmos

1.3 | BUSINESS INVESTITION IN DEVOPS

DevOps ist eine weitreichende Investition in Menschen, Prozesse und Tools wodurch sich Herausforderungen stellen, denen üblicherweise mit Methoden des **Change Managements** begegnet wird. In personeller Hinsicht müssen die Arbeitskräfte auf ein breiteres, **fachübergreifendes Aufgabengebiet** vorbereitet werden und sich offen für neue Prozesse und Technologien zeigen.

Beim Erarbeiten der Prozesse wird die Umstellung von formalen Freigabestrukturen auf automatisierten Tests und Deployments, Aufwand in Anspruch nehmen. Zudem verändert sich der **Arbeitsrhythmus und die Arbeitskultur**. Darüber hinaus muss der technologische Status Quo ermittelt werden, um eine Aussage treffen zu können, inwieweit die vorhandenen Systeme und Organisationsprozesse DevOps-kompatibel sind und speziell dem erhöhten Automatisierungsgrad Rechnung tragen.

Im Hinblick auf neue Tools kann auf eine Reihe von **Open-Source-Produkten** zurückgegriffen werden. Allerdings können sich auch hier Aufwände ergeben. Zur Bewältigung dieser Her-

-ausforderungen ist die Begleitung durch erfahrene Experten ratsam. Diese können in der Phase der Transformation Anpassungsschwierigkeiten abfedern und dabei helfen, dass die Geschwindigkeits- und Qualitätsvorteile schnell zum Tragen kommen.

1.4 | BUSINESS MEHRWERT DURCH DEVOPS

Aus wirtschaftlicher Sicht weckt DevOps zwei zentrale Erwartungen: Die erste besteht darin, konzeptionelle Ideen schneller in Produktion zu bringen und damit früher wertstiftenden Output zu generieren. Wichtige Parameter in diesem Zusammenhang sind **Durchlauf- und Verarbeitungszeit** (Abb. 1.5).

Die Relation beider Kennzahlen gilt als wichtiger Indikator für die Effizienz des Wertstroms. DevOps setzt genau an diesem Punkt an und versucht insbesondere die Zeit, die eine Entwicklung in der Warteschlange zum Deployment verbringt, zu reduzieren und damit die **Time-to-Market** zu beschleunigen.

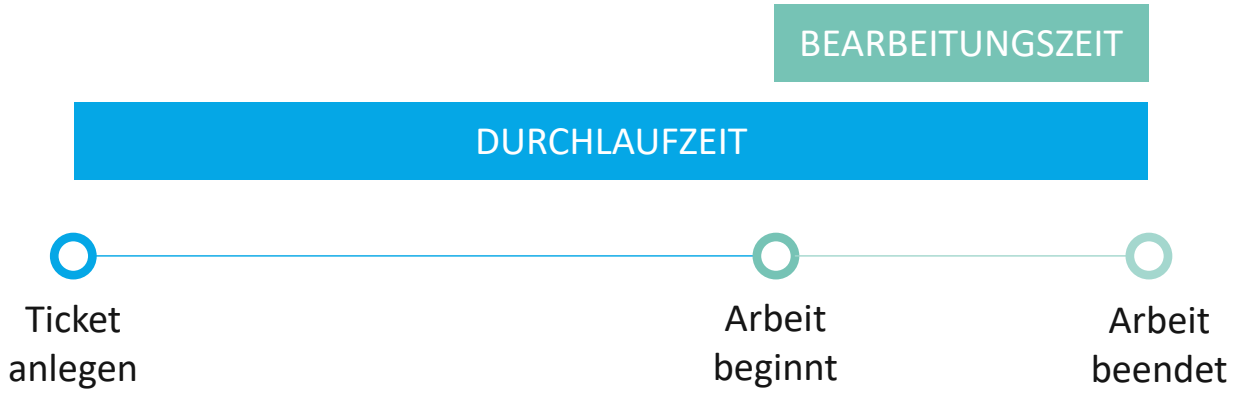

Abb. 1.5: Durchlauf- u. Verarbeitungszeit

Der Schlüssel hierzu sind kleine Entwicklungspakete, die Entwickler schnell und voneinander unabhängig integrieren, validieren und letztlich ausliefern können. In diesem Merkmal liegt zudem der zweite wesentliche wirtschaftliche Vorteil, der sich durch die **Risikominimierung** im Vergleich zum Bau komplexer, monolithischer Entwicklungen ergibt (Abb. 1.6).

Probleme im Zusammenspiel verschiedener Komponenten werden früher erkannt und sind einfacher nachvollziehbar und behebbar. Insgesamt sinkt so die **Time-to-Value** und gleichzeitig steigt die Reaktionsfähigkeit und damit das Vermögen Kundenbedürfnisse besser zu verstehen.

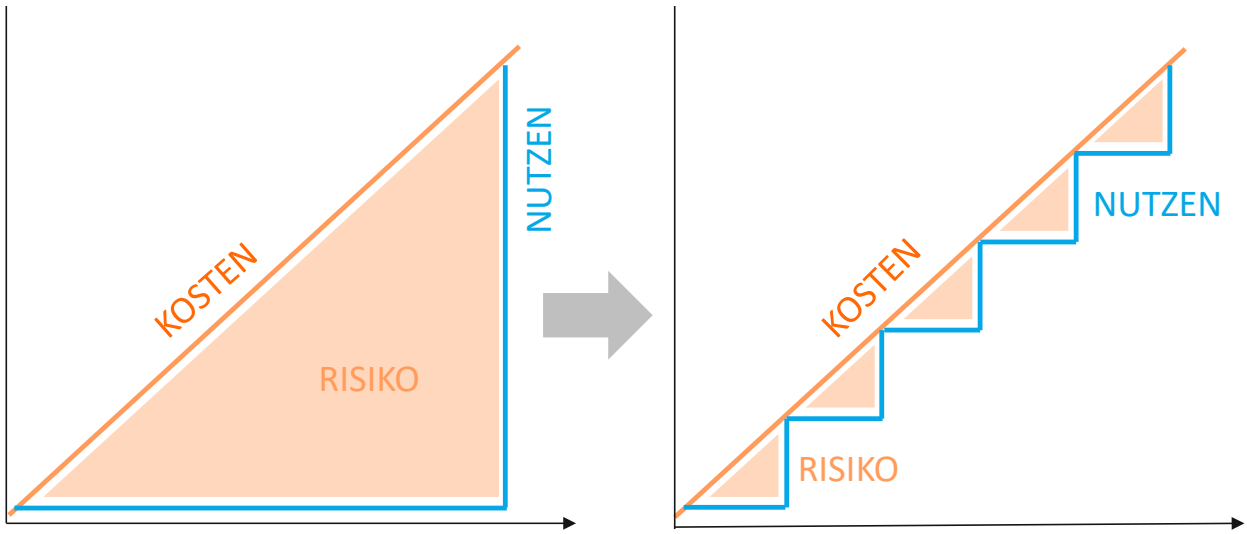

Abb. 1.6: Risikominimierung durch iteratives und inkrementelles Vorgehen

Insoweit wird auch von einer Verschiebung vom Projektmanagement zum Produktmanagement gesprochen, in dem sich eine Ende-zu-Ende-Verantwortung der beteiligten Personen entlang des Wertstroms ergibt. Dies bedeutet, dass sich die Arbeit auf die Validierung von Produktideen fokussiert, was durch die kurzen Durchlaufschleifen und die aktive Feedbackkultur gewährleistet wird.

Damit liefert DevOps auch wesentliche strategische Mehrwerte. In der Softwareentwicklung wird der Ansatz immer beliebter und die erfolgreiche Adaption dieser agilen Philosophie wird auf Kundenseite als Qualitätsmerkmal honoriert. Dies gilt umso mehr bezüglich des Wettbewerbs um fähige und engagierte Mitarbeiter, die entsprechende Arbeitsbedingungen zunehmend erwarten.

1.5 | DEVOPS AUF EINEN BLICK

Zusammenfassend lässt sich DevOps wie folgt beschreiben:

ZIELE

Ziel von DevOps ist es, durch gemeinsame Anreize, Prozesse und Software-Werkzeuge eine effektivere und effizientere Zusammenarbeit der Bereiche Entwicklung und Betrieb zu ermöglichen und dadurch

- Die Durchlaufzeit von wertstiftenden Entwicklungszyklen zu vermindern (Time-to-Market, Abb. 1.5)
- Die Qualität der Auslieferungen zu erhöhen (Time-to-Value, Abb. 1.6)
- Die Risiken von schwerwiegenden Fehlern und Problemen zu minimieren (Abb. 1.6)

CHARAKTERISTIKA

- Kleine Entwicklungen und Auslieferung lauffähiger Artefakte (CI/CD)
- Integrierte, Tool-gestützte und weitgehend automatisierte Entwicklungsprozesse (CI)
- Umfangreiche und weitgehend automatisierte Qualitätskontrollen (CT)
- Zusammenlegung von Verantwortlichkeiten und Berechtigungen (CT, Ende-zu-Ende)
- Häufige Wiederholung der kleinen Durchlaufzyklen (CD)
- Enge und häufige Kommunikation aller beteiligten Stellen und Systeme (Continuous Feedback)

HERAUSFORDERUNGEN

- DevOps ist eine umfangreiche Investition in Menschen, Prozesse und Technologie und eine Philosophie, die von den Beteiligten gelebt werden muss
- In der Transformationsphase sind Anpassungsschwierigkeiten zu erwarten, die die Geschwindigkeits- und Qualitätsvorteile negativ beeinträchtigen können
- Das größte Risiko besteht darin, die Menschen im Rahmen der Transformation zu verlieren

VORTEILE

- Die Durchlaufzeit des Wertstroms sinkt und es werden nutzbare Ergebnisse in kurzen Zeiträumen erzielt
- Durch die kürzeren Durchlaufzyklen und Feedbackschleifen steigt die Identifikation der eingebundenen Anwender mit ihrem Produkt und verstärkt den positiven Effekt
- Durch die kontinuierlichen Feedbackschleifen steigt die Qualität der Auslieferungen und die Steuerung von Veränderungen, insbesondere durch neue Markt- oder Umweltbedingungen, wird leichter
- Das Risiko schwerwiegender Fehler beim Deployment in der Produktivumgebung wird erheblich minimiert
- Die organisatorische Komplexität der IT-Infrastruktur nimmt durch die engere Verknüpfung der Fachbereiche Entwicklung und Betrieb ab

2 | DEVOPS IM DATA-WAREHOUSE-KONTEXT

Aufgrund der skizzierten Vorteile hat sich DevOps im Bereich der Softwareentwicklung als präferierter Ansatz bereits weitgehend etabliert. Nach über 10 Ausgaben taucht DevOps beispielsweise seit 2017 kontinuierlich im jährlich erhobenen **State Of Agile Report** auf und zeigt Umfragewerte nach denen über 70% der befragten Unternehmen in DevOps investieren.

Im Data-Warehouse-Kontext hingegen befindet sich die Einführung von DevOps-Praktiken und -Werkzeugen noch im Anfangsstadium. Das Thema nimmt jedoch in den letzten Jahren stark an Fahrt auf. Trends, wie **DataOps** basieren vorwiegend auf DevOps-Prinzipien und verfolgen das Ziel, die gesamte Wertschöpfungskette des Datenmanagements besser zu integrieren. Getrieben wird diese Entwicklung insbesondere durch ein vielschichtiges, neues Anforderungsprofil an das Datenmanagement und die Konzeption eines Data Warehouse.

Weitere Informationen dazu finden Sie in unserem Whitepaper "**7 BI-Trends, die Sie kennen sollten**".

2.1 | ANFORDERUNGEN AN DAS ZUKUNFTSFÄHIGE DATA WAREHOUSE

Der Hauptfaktor, der zu einem Modernisierungsbedarf der bestehenden Data-Warehouse-Strukturen führt, ist das erhöhte Interesse von Business-Entscheidern an datenbasierten Insights. Dies beinhaltet sowohl die Möglichkeit zu dynamischen Anpassungen vergangenheitsorientierter BI-Reports als auch die Durchführung zukunftsgerichteter Analysen durch Maschinelles Lernen und Künstliche Intelligenz.

All diese Anwendungsfälle erfordern unterschiedliche Methoden der Datengewinnung, -aggregierung und -aufbereitung, wodurch sich eine größere Notwendigkeit an Flexibilität hinsichtlich der Datenströme durch das Data Warehouse ergibt.

Der Umstand der erhöhten Reporting- und Analytics-Nachfrage beruht vor allem auf der wachsenden Verfügbarkeit immenser Datenmengen, insbesondere durch neue Technologien wie das Internet of Things (IoT) und Social Media. Diese Datenmengen müssen in die Data-

Warehouse-Landschaft eingebunden werden, und zwar so, dass sie jederzeit auswertbar sind. Dabei steigt nicht nur die Datenmenge, sondern auch die Datenvielfalt.

Strukturierte und unstrukturierte Daten müssen im Data-Warehouse-Kontext zusammengeführt werden. Dazu bedarf es der Integration spezialisierter Systeme, so dass sich das Bild einer heterogenen Data-Warehouse-Landschaft ergibt. Dessen Hauptaufgabe liegt damit neben der Persistierung der Daten speziell in der Orchestrierung der komplexen Datenkommunikationsprozesse (Abb. 2.1).

Entscheidende Faktoren in dieser Gemengelage sind Flexibilität und Agilität, wodurch DevOps als prozessuales Rückgrat der stetigen Data-Warehouse-Entwicklung in den Fokus rückt und entsprechende Erfahrungen der Softwareentwicklung auf den klassischen Ansatz einer Data-Warehouse-Entwicklung übertragen werden.

Dabei sind auch die geschilderten strukturellen DevOps-Vorteile, wie die Vereinfachung der IT-Organisation, mit übergreifenden Verantwortlichkeiten und einheitlichen Vorgehensweisen, Motivation DevOps zu nutzen.

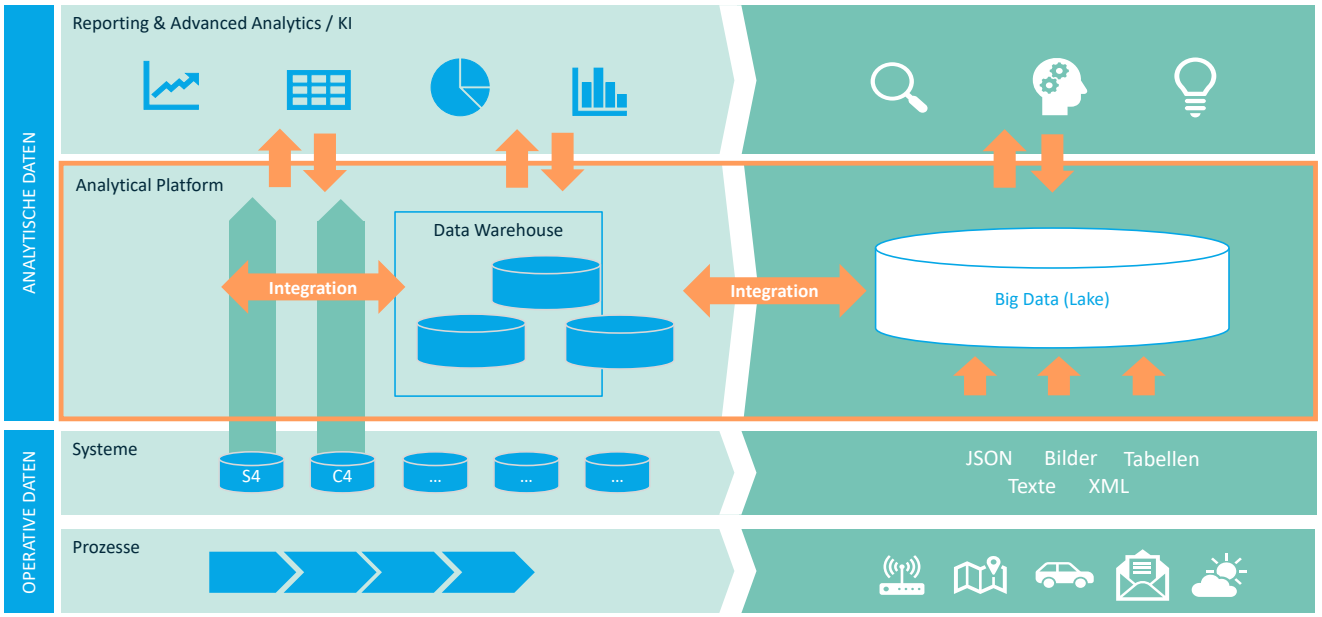

Abb. 2.1: Erfordernis einer flexiblen und agilen DWH-Landschaft

2.2 | KLASSISCHE VS. DEVOPS-DWH-ENTWICKLUNG

In der klassischen Data-Warehouse-Entwicklung teilen sich die Entwickler den gleichen Arbeitsbereich und eine Laufzeit und arbeiten an der gleichen Version direkt auf der Datenbank (Abb. 3.2). Dies führt zu folgenden potenziellen Konflikten:

Die Abhängigkeiten in Bezug auf Daten und Objekte sind im Rahmen paralleler Arbeit schwer zu managen. Veränderungen einer Person wirken sich unmittelbar auf die Parallelentwicklungen der Kollegen aus. Hierdurch ist es kaum möglich neue Funktionalität isoliert zu entwickeln und zu testen.

Den Entwicklern wird in diesem Konstrukt die Möglichkeit geraubt, Szenarien durchzuspielen und auszuprobieren, da dies aufgrund der Abhängigkeiten in der gemeinsamen Umgebung leicht zu Verwirrung und Fehlern führt. Das Ausweichen auf lokale Umgebungen ist aufgrund der großen Datenmengen sowie der Abhängigkeit von externen Serververbindungen häufig keine Option.

Bugfixes für Fehler, die sich erst nach Auslieferung in die Produktivumgebung zeigen, sind problematisch. Korrekturen sind aufwendig, wenn sich die Entwicklungsumgebung in der Zeit des Auslieferungsprozesses verändert hat und die Entwicklungsumgebung für den Fix zunächst auf den Stand des Produktivsystems zurückgesetzt werden muss.

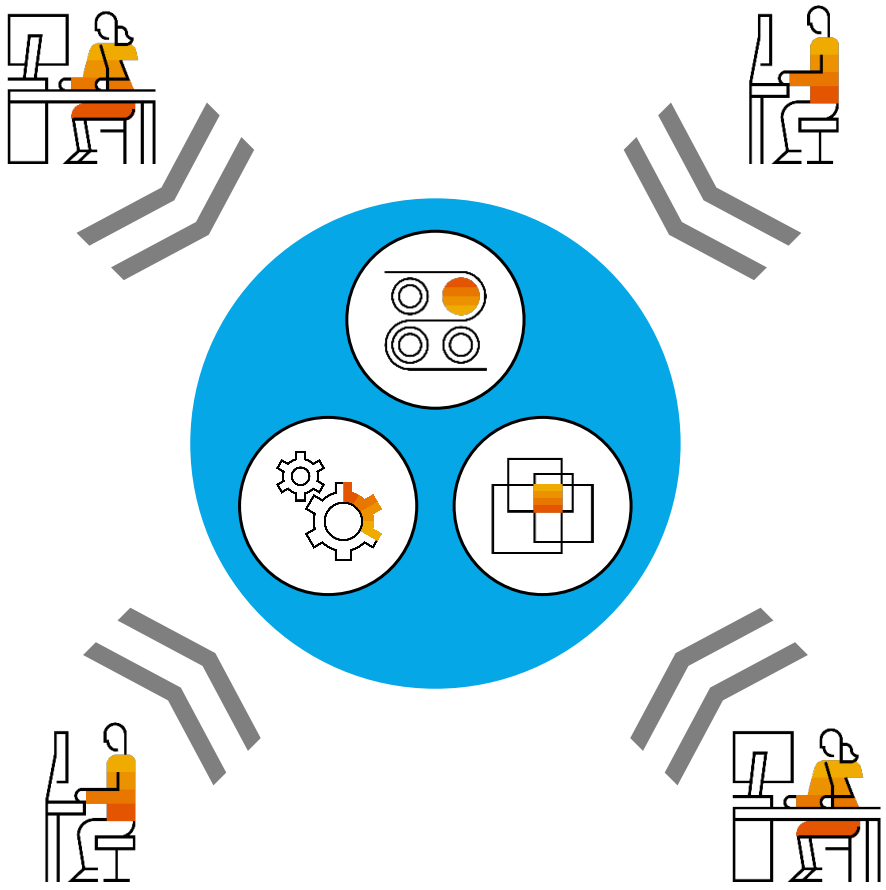

Abb. 2.2: Klassische DWH-Entwicklung

Der Schlüssel zur Umgehung dieser bekannten Problemstellungen liegt in der Verfügbarkeit isolierter Arbeitsbereiche, die als Sandboxes bezeichnet werden. Diese Sandboxen gewährleisten, dass Applikationen und Entwicklungen vollkommen isoliert voneinander ablaufen können und somit ein komplett eigenständiger Entwicklungsbereich für jeden Entwickler zur Verfügung steht. In diesen Bereichen können unterschiedliche Systemruntimes, Programmiersprachen und eigene Versionen von Entwicklungen erarbeitet und getestet werden. Somit können Entwicklungsarbeiten leicht, parallel durchgeführt werden. Die Zusammenarbeit erfolgt im Anschluss über ein gemeinsames Source Code Repository, wie Git (Abb. 2.3).

Soweit entspricht dies einer einfachen Adaption des in 1.2.1 »Continuous Integration« beschriebenen Continuous-Integration-Prozesses. Der erfahrene DWH-Entwickler wird sich an dieser Stelle jedoch vermutlich fragen, wie eine solche Arbeitsweise in Bezug auf die im DWH enthaltenen Daten, die verschiedenen Quellsysteme und Integrationslayer Sinn ergeben kann. Die Antwort liegt in einer angepassten Data-Warehouse-Architektur, die die Trennung von Daten- und Business-Logik erlaubt. Möglich wird dies durch den Einsatz von Datenvirtualisierungen und der Abstraktion von Datenbankobjekten.

Hierzu wird die Form der deklarativen Programmierung genutzt, die sich auf Zielbeschreibungen, also die Frage Was erreicht werden soll, konzentriert. Eine Art Compiler setzt die Anweisungen dann in entsprechende Befehle für die Datenbank um und erweitert oder verändert bestehende Objekte entsprechend der Zielbeschreibung.

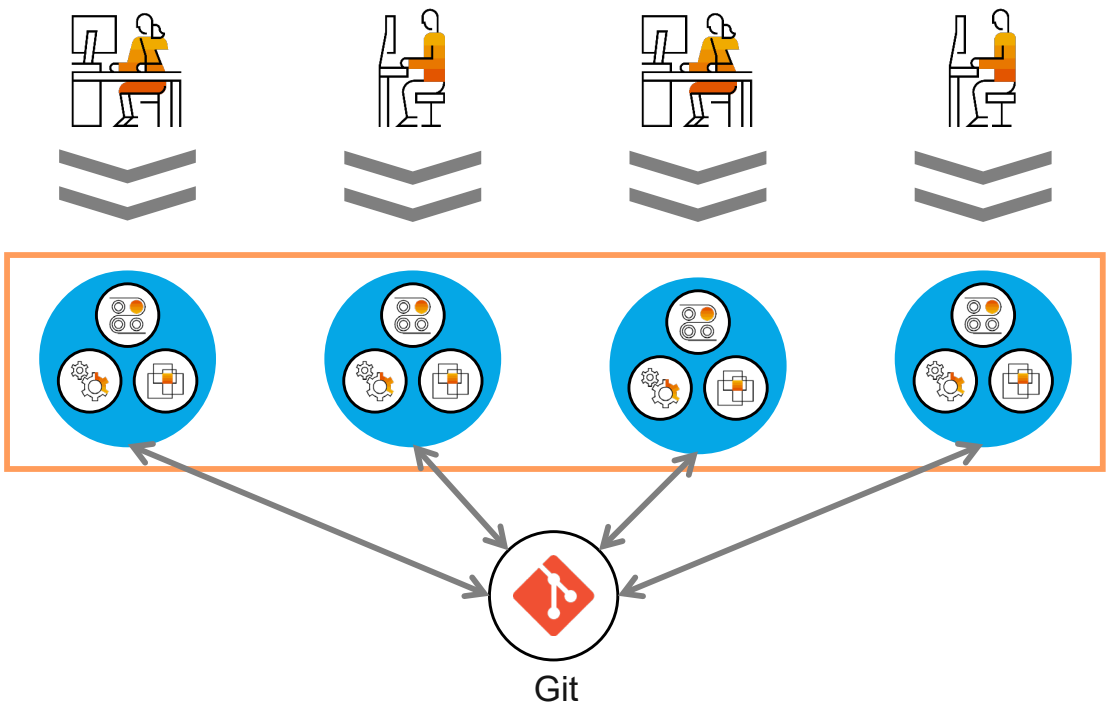

Abb. 2.3: Sandbox-basierte DWH-Entwicklung

2.3 | VERTEILTE SOURCE-CODE-VERWALTUNG

Notwendige Bedingung für einen Sandbox-basierten Entwicklungsansatz ist ein gemeinsames Source Code Repository. Analog zum Entwicklungsansatz handelt es sich auch hierbei um ein verteiltes System (Abb. 2.5), in dem eine Version des gesamten Codes für jeden Entwickler vorgehalten wird und im Gegensatz zur zentralen Versionskontrolle (Abb. 2.4) nicht direkt in das Hauptverzeichnis geschrieben wird.

Die geklonten Repositories werden durch sogenannte „pull“ und „push“ mit dem Hauptverzeichnis synchronisiert und jeder Entwickler kann anschließende auf seinen Klon zurückgreifen (Abb. 2.5).

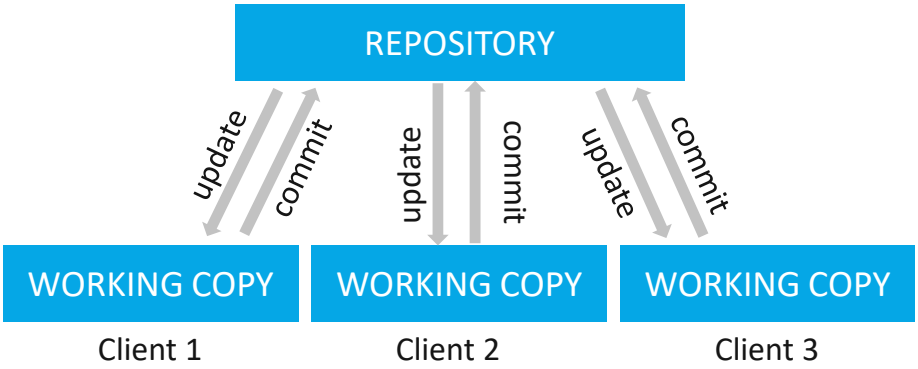

Abb. 2.4: Zentrale Versionskontrolle

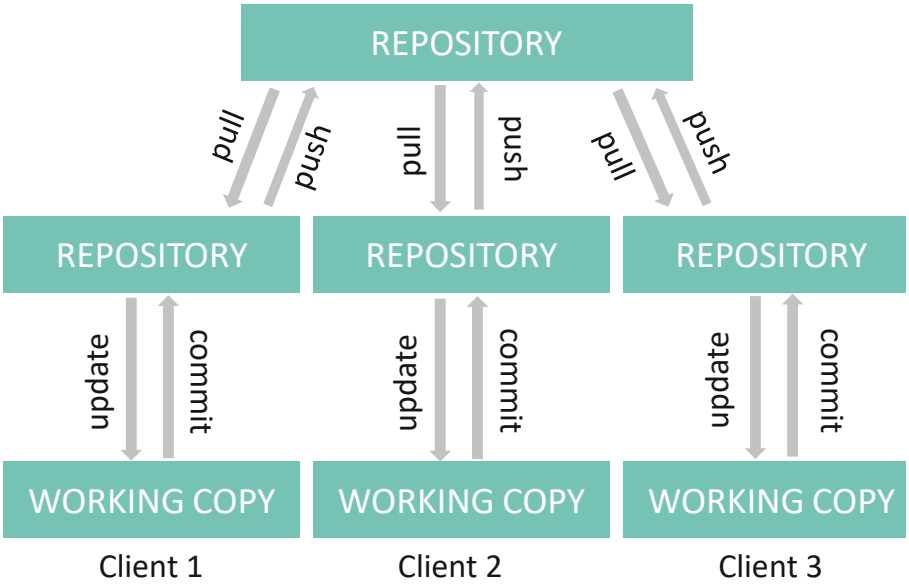

Abb. 2.5: Verteilte Versionskontrolle

Hierdurch bieten sich die zuvor beschrieben Arbeitsvorteile, da das Aufteilen und Zusammenfügen von Codebestandteilen wesentlich erleichtert wird. Der bekannteste Vertreter dieser Art von Version Control System (VCS), ist das von Linus Torvald entwickelte Git. Git folgt dem Konzept des „Branching“, das versucht für jedes Entwicklungsfeature einen eigenen Zweig und damit ein Repository zu generieren, auf dem gearbeitet wird. Hinzukommen Qualitätsicherungs-Branches, wie Release und Development. Die so entstandenen Verzweigungen / Repositories werden durch „push“ und „pull“ im Master-Branch der Produktivumgebung wieder zusammengefügt (Abb. 2.6).

Die Git-Technologie wird von verschiedenen Dienstleistern, wie GitHub, GitLab oder BitBucket als Cloud- oder On-Premise-Lösung angeboten und muss lediglich in die Entwicklungsumgebung eines Data Warehouse eingebunden werden. Aufgrund dieser technischen Voraussetzungen im Hinblick auf Sandbox-fähige Strukturen und die Möglichkeit der Einbindung gängiger DevOps-Tools, insbesondere einer verteilten Source Code Verwaltung, sind die Anbieter von Data-Warehouse-Lösungen aufgefordert entsprechende Maßnahmen zu treffen.

Dieser Markt entwickelt sich erst langsam, erste DevOps unterstützende Produkte sind jedoch bereits verfügbar.

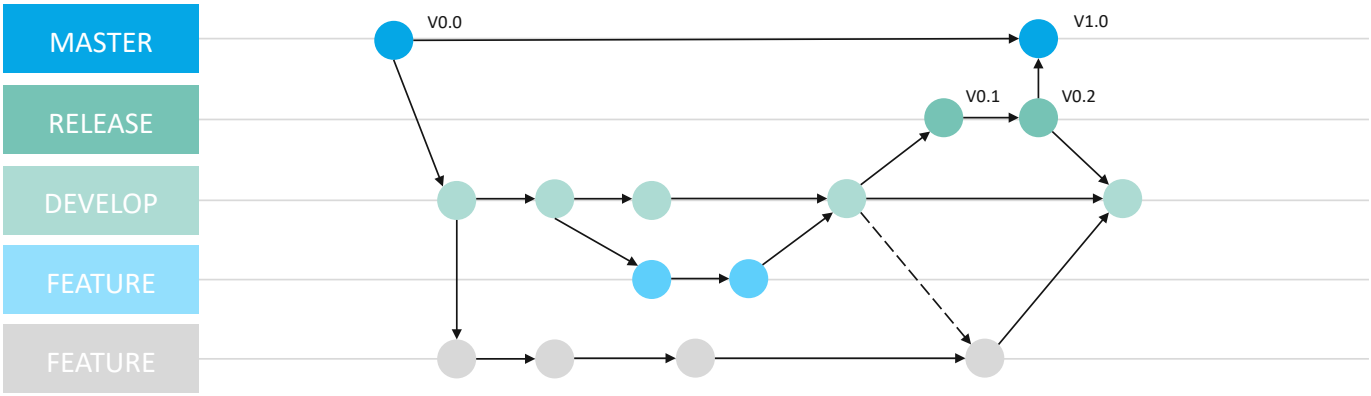

Abb. 2.6: Git Branching

2.4 | SAP SQL DATA WAREHOUSING

Der entsprechende Ansatz aus dem Hause SAP ist das SAP SQL Data Warehousing. Im Gegensatz zur bisher anwendungsgetriebenen Vorgehensweise im Rahmen des SAP BW/4HANA, öffnet sich der Softwareriese aus Walldorf mit diesem Ansatz mehr als je zuvor und schafft Raum für flexible und individuelle Data-Warehouse-Lösungen auf Basis der HANA-Plattform.

Eine wesentliche Komponente des erhöhten Freiheitsgrades ist dabei die Ermöglichung von DevOps-Auslieferungszyklen im Rahmen der Data-Warehouse-Implementierung und Weiterentwicklung. Dazu abstrahiert SAP die Entwicklungsarbeit durch einen integrierte Anwendungsplattform (SAP HANA Extended Application Services Advanced Model – kurz XSA) von der HANA-Datenbank und erlaubt eine Sandbox-basierte Gliederung des Prozesses in Designtime und Runtime.

Die XSA-Plattform stellt Entwicklungstools zur Verfügung und gestattet die Anbindung aller gängiger (Open Source) DevOps-Tools. Insbesondere die Verbindung zu einem Git-Repository ist standardmäßig vorgesehen, so dass Entwickler in ihrer eigenen Sandbox arbeiten und mithilfe von Git Artefakte miteinander integrieren und ausliefern können (Abb. 2.7).

Durch diese Konstruktion ergibt sich eine dezidierte DevOps-Entwicklungsmethode, deren Phasen, Prozesse und Tools dem klassischen Bild der Softwareentwicklung ähneln.

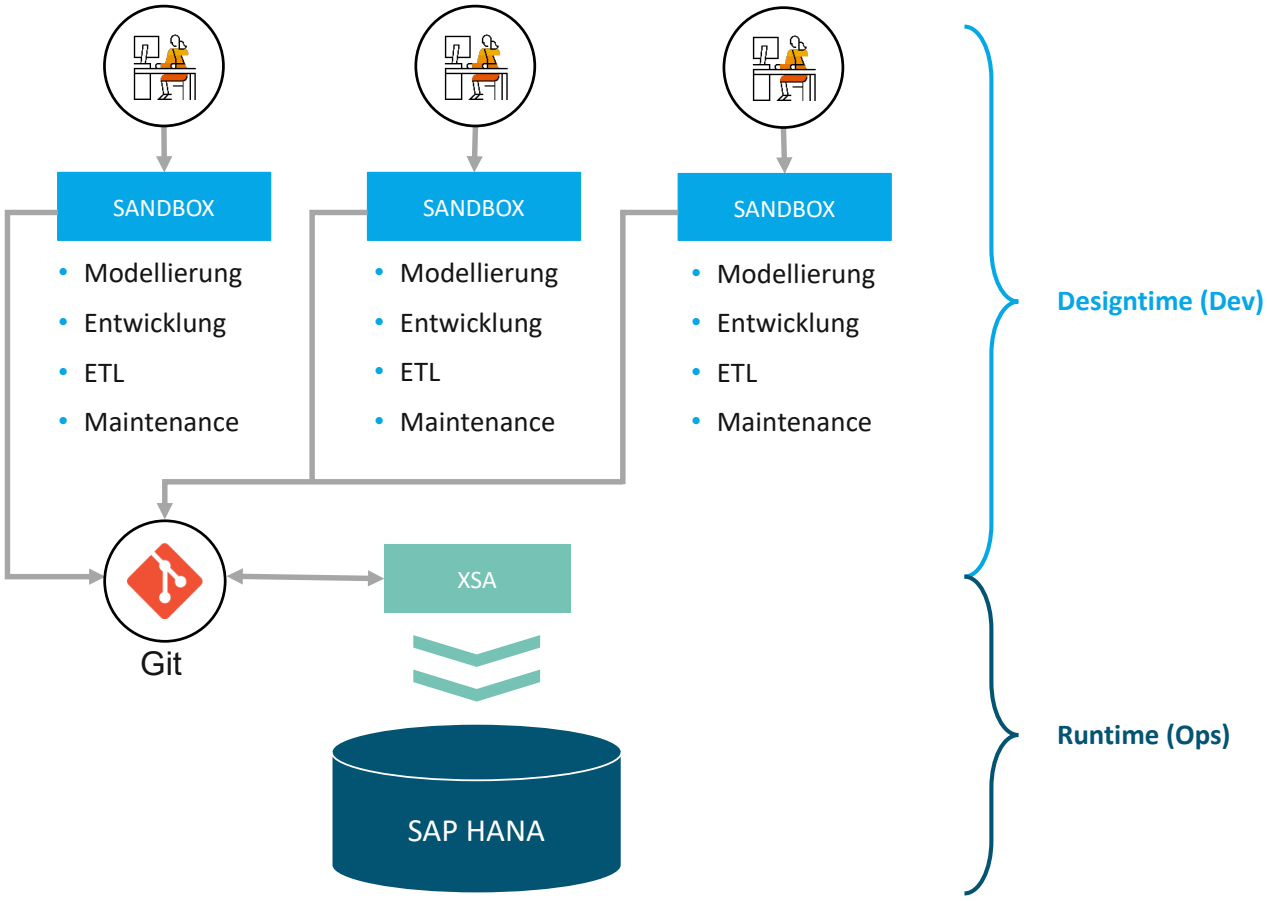

Abb. 2.7: Git Branching

2.4.1 | CONTINUOUS INTEGRATION

Die kontinuierliche Integration wird durch die standardmäßige Anbindung eines Git-Repositories an den Anwendungsserver XSA sichergestellt. Die Anforderungen an das Data Warehouse werden von den Entwicklern parallel erarbeitet. Nach einem initialen „pull“ des Gesamtstandes der Entwicklung, arbeiten die Beteiligten individuell und verschmelzen entweder in ihrer Sandbox den aktuellen Stand der Gesamtentwicklung oder bringen ihren persönlichen Entwicklungsstand in die Gesamtaufgabe ein (Abb. 2.8).

Das Git-Repository versioniert alle Veränderungen auf jeglichen Branches und ermöglicht übersichtliche Gegenüberstellungen von Versionen, so dass die Fehlersuche sowie das Hin- und Herspringen zwischen Versionsständen leicht möglich sind.

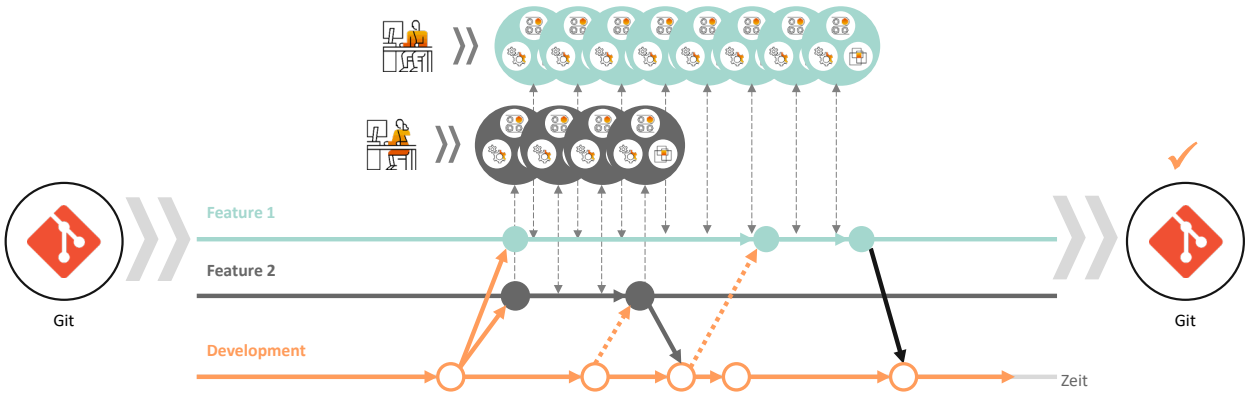

Abb. 2.8: Continuous Integration der Entwicklungen

2.4.2 | CONTINUOUS DELIVERY

Im Rahmen der kontinuierlichen Auslieferung setzt sich die CI-Arbeitsform konsequent fort. Über dem Entwicklungsstand gibt es im Repository die weiteren Branches „Release“ und „Master“. Der beschriebene Akt der Verschmelzung von Versionsständen findet nun auch auf diesen Ebenen statt, mit dem Ziel Veränderungen möglichst schnell in die Produktivumgebung und zu den Anwendern in den Fachabteilungen zu bringen (Abb. 2.9).

In diese Prozesse werden daher zusätzlich Automatisierungstools wie Jenkins oder Bamboo integriert. Diese reagieren auf neue Versionsstände auf den einzelnen Branches der Auslieferungspipeline im Git und initiieren automatisch eine Veränderung auf die nächsthöhere Ebene.

Dabei werden in der Regel auch zuvor definierte Tests, beispielsweise auf Namenskonventionen, automatisch durchgeführt. Wird die Pipeline soweit perfektioniert, dass Veränderungen automatisch alle Branches durchlaufen und bei Fehlerfreiheit unmittelbar produktiv gehen, wird von Continuous Deployment gesprochen.

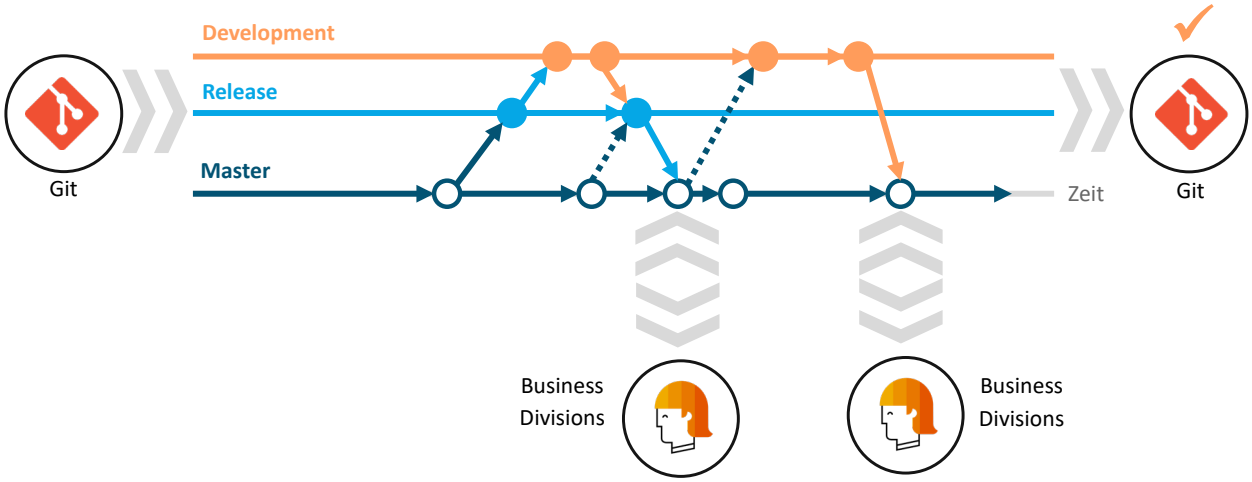

Abb. 2.9: Coninuous Delivery der Entwicklungen

2.4.3 | CONTINUOUS TESTING

Für ein erfolgreiches CI/CD ist das kontinuierliche und im besten Falle automatisierte Testen von entscheidender Bedeutung. Hierzu können die speziellen CI/CD-Test-Tools der DevOps-Softwareentwicklung genutzt werden. Diese werden, wie beschrieben, bei einer Verschmelzung durch die Versionierung im Git aktiviert und prüfen die Qualität der neuen Codebestandteile automatisch durch Unit, Acceptance, Regression und Performance Tests (Abb. 2.10). Die Automatisierung verläuft grundsätzlich Bottom-Up, d.h. kritische Verschmelzungen auf Master- oder Release-Ebene werden zunächst weiterhin manuell ausgeführt oder unterstützt. Wird auch in diesem Bereich der Grad manueller Arbeit eliminiert, kann ein Continuous Deployment erfolgen.

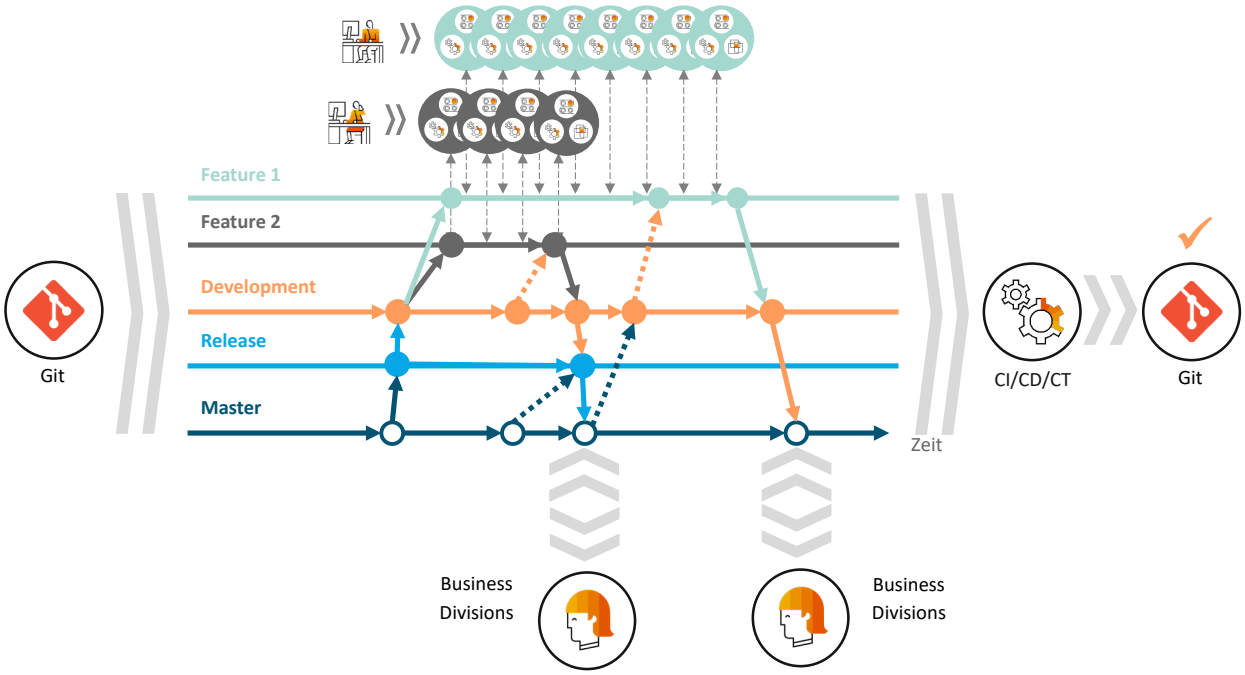

Abb 2.10: Continuous Testing durch automatisierte CI/CD-Tests

2.4.4 | CONTINUOUS FEEDBACK

Durch CI/CD in Verbindung mit dem kontinuierlichen Testen und den kurzen Auslieferungs- bzw. Deployment-Zyklen ergibt sich im besten Fall konstantes Feedback über alle Phasen und Prozesse hinweg (Abb. 2.11). Es ist wichtig, dass dieses Feedback, insbesondere das der Nutzer in den Fachbereichen, tatsächlich wieder in den DevOps-Zyklus einfließt. Für diese Aufgabe haben sich Tools zum Issue Tracking, wie beispielsweise Jira, bewährt.

Diese können über das Git-Repository direkt in die Deploymentpipelines integriert werden. Hierdurch lassen sich automatische Statusberichte auf Basis der Source-Code-Veränderungen leicht mit manuell erfasstem Feedback an einem Ort zusammenfassen und machen ein kontinuierliches Anforderungs- und Änderungsmanagement möglich.

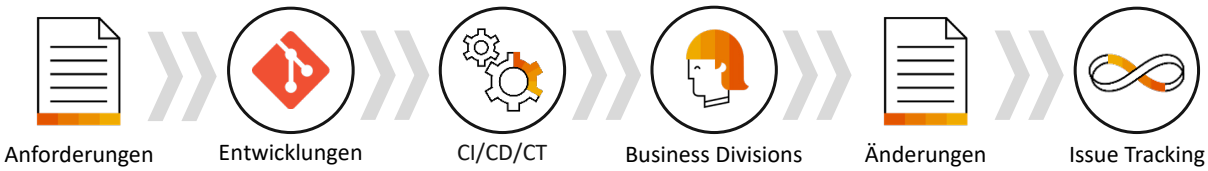

Abb 2.11: Continuous Feedback

2.4.5 | DEVOPS-PHASEN

Für die Entwicklung des SAP SQL Data Warehouse ergeben sich somit ebenfalls acht Phasen (Abb. 2.12). Im Vergleich zur Softwareentwicklung liegt der Fokus insgesamt nicht so stark auf dem Coding, sondern auf der Konzeptionierung der Datenmodelle und der Modellierung der nachgelagerten Datenströme durch das Data Warehouse. Die Abstraktion von der HANA-Datenbank durch den XSA-Server macht es möglich, dass mit graphischen Tools gearbeitet und weitgehend auf SQL-Code verzichten werden kann. Dieser wird effizient im Hintergrund erzeugt, um der modellierten Zieldefinition zu entsprechen. Es ist wichtig zu verstehen, dass alle Phasen für jeden sukzessiven Schritt der DWH-Entwicklung Anwendung finden. Aufgrund der Flexibilität des SAP-SQL-DWH-Ansatzes kommt der Datenmodellierung eine große Bedeutung zu. Die einzelnen Phasen lassen sich wie folgt charakterisieren:

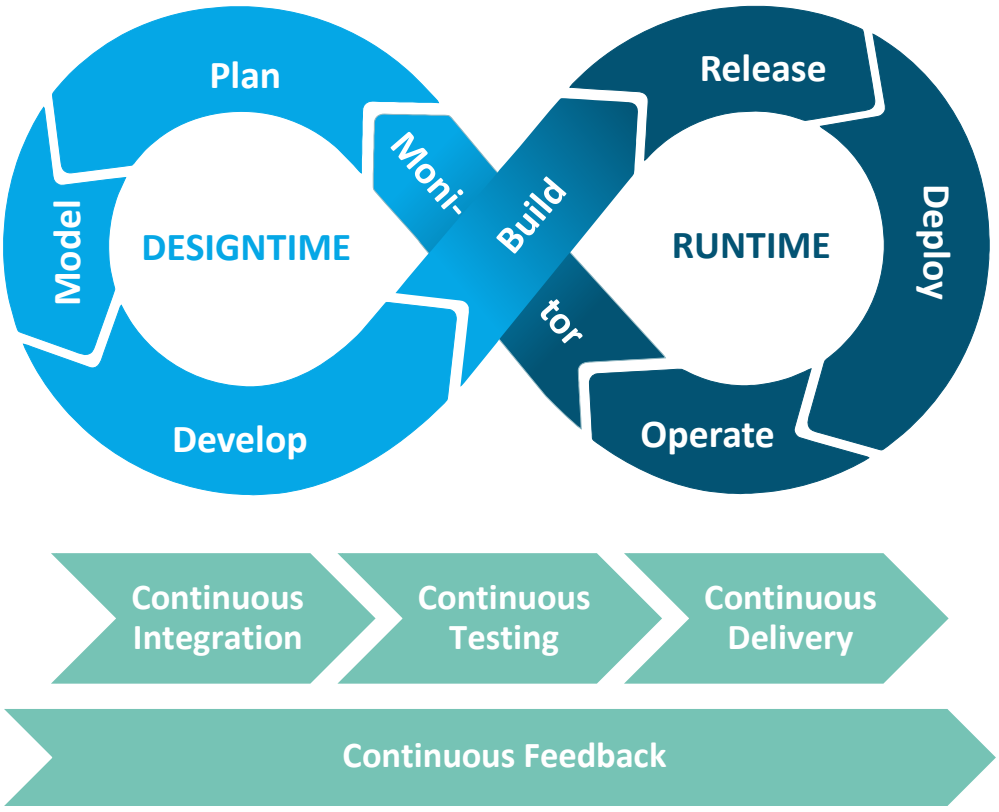

Abb. 2.12: DevOps-Phasen der SAP-HANA-SQL-Data-Warehouse-Entwicklung

1. Plan: Am Anfang stehen die Anforderungen der Anwender an das Data Warehouse. Diese müssen konzeptionell aufgearbeitet werden, um Modelle anfertigen zu können.

2. Model: Auf Basis der Konzepte werden die verschiedenen Artefakte graphisch modelliert. Hierzu gehören Datenmodelle, Datenstrom-Logiken und DatenAnsichten. Auch wenn diese im Gesamtprozess aufeinander aufbauen, wird im DevOps-Zyklus die Auslieferung jedes Artefakts vollständig vollzogen, so dass sich ein iterativer und inkrementeller Prozess ergibt.

3. Develop: Die Codierung wird zumeist im Hintergrund durch die XSA-Plattform geleistet. Es ist jedoch auch direktes Entwickeln mit SQL-Code möglich.

4. Build: Durch sogenannte „Builds“ werden die Designtime-Artefakte in die Runtime überführt. Durch die Sandboxes kann dies zunächst isoliert erfolgen. Über die verschieden Branches im Git-Repository erfolgt im Anschluss die weitere Auslieferung, die mit automatischen Tests einhergeht.

5. Release: Auf der nächsten Stufe werden etwaige Entwicklungsstände in den Auslieferungszustand gebracht. Auch hier sichern automatisierte Tests ungehinderte Abläufe.

6. Deploy: Im Anschluss erfolgt die Integration ins Produktivsystem. Ziel von DevOps ist das automatisierte Deployment kleinster Entwicklungsartefakte.

7. Operate: Die Anwender arbeiten operativ mit der Neuentwicklung.

8. Monitor: Die finale Überprüfung wird von den spezifischen Nutzern vorgenommen und Feedback fließt an die Entwickler zurück.

2.4.6 | TOOLS

Im Hinblick auf die verwendbaren Tools ist der Anwender im Rahmen eines SAP SQL Data Warehouses flexibel (Abb. 2.13). Die XSA-Platform bringt zum Modellieren und Entwickeln die Applikation SAP Web IDE mit. Zudem kann der SAP PowerDesigner zum Planen und Modellieren sehr gut genutzt werden. Die Git-Anbindung ist standardmäßig vorgesehen. Der größte Spielraum ergibt sich im Hinblick auf die CI-/CT-/CD-Werkzeuge und -Automatisierungstools. Hier sind beispielsweise Jenkins und Bamboo zu nennen. Auch beim Issue Tracking ist das Angebot groß. Tools wie Jira oder Confluence setzen hier aktuell die Benchmark.

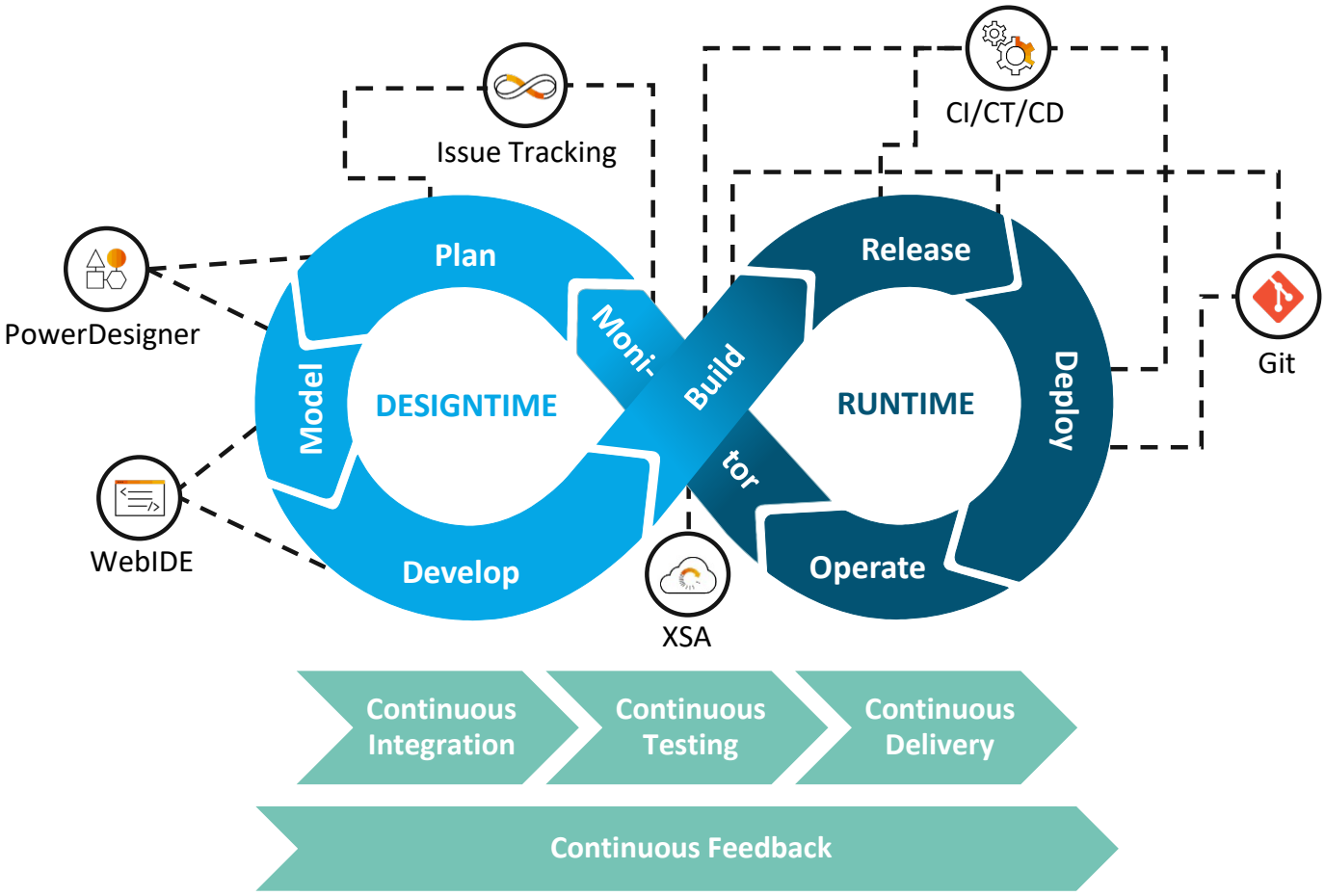

Abb. 2.13: DevOps-Phasen der SAP-HANA-SQL-Data-Warehouse-Entwicklung

2.4.7 | INVESTITION SAP SQL DATA WAREHOUSING

Neben den klassischen Anforderungen einer DevOps-Transformation, die bereits in 1.3 beschrieben wurden und ein adäquates Change Management erfordern, ist für das SAP SQL Data Warehousing natürlich die Investition in entsprechende Hard- und Software nötig. Zentrales Element ist hier SAP HANA 2.0, die als Plattform den integrierten Application Server XSA mit weiteren Softwarelösungen, wie der SAP Web IDE mitbringt. Da es sich bei SAP HANA um eine In-Memory Datenbank handelt, müssen für den On-Premise-Einsatz entsprechende Serverkapazitäten, speziell im Hinblick auf Arbeitsspeicher, vorhanden sein. Eine Alternative

bietet die Cloudversion. Daneben ist die Nutzung des SAP PowerDesigners zu empfehlen. Dieses Tool muss zwar separat erworben werden, bietet aber die beste Unterstützung für die Modellierung. Für den Start ist man mit dieser Ausgangsbasis bestens gerüstet.

Die Git-Anbindung ist in der WebIDE bereits vorgesehen. Ein Git-Server wird jedoch noch benötigt. Hier sollten sich allerdings Synergien mit anderen Bereichen in der IT ergeben, die gegebenenfallsbereits mit der Git-Technologie arbeiten. Gleiches gilt für einen zentralen Build-Server, der für Continuous Integration benötigt wird, sowie für Issue-Tracking-Lösungen, die sich für ein erfolgreiche Projektmanagement als Standard etabliert haben. Die Automatisierung von CI/CD und Continuous Testing lässt sich zudem gut mit Open Source Tools abbil-den, so dass sich auch hier zunächst kein er-höhter Investitionsbedarf ergibt.

Es bleibt zu betonen, dass die entscheiden-de Hürde viel mehr in einem erfolgreichen Change-Management in Bezug auf die Faktoren Menschen, Prozesse und Technologien liegt. Insbesondere ersterem Faktor ist dabei das nö-tige Maß an Aufmerksamkeit zu schenken.

Die Bereitschaft zur Adoption der DevOps-Philosophie ist unerlässlich und setzt mehr voraus als notwendige fachliche Kriterien, wie ein so-lides SQL-Knowhow und Erfahrungen im Data-Warehouse-Umfeld.

2.4.8 | VORTEILE DES SAP SQL DATA WAREHOUSING

Im Rahmen des SAP SQL Data Warehousing kommen die Vorteile des DevOps-Entwi-cklungsansatzes voll zum Tragen. Durch die vorhandenen Sandboxes und das gemeinsame Git-Repository wird ein unabhängiges, paral-leles Arbeiten möglich und der Weg zu einem iterativen und inkrementellen DevOps-Entwicklungsprozess geebnet.

Der modulare Aufbau des SAP SQL Data Warehousing und der Schwerpunkt auf Modellierung statt Codierung erlauben einen klaren achtstufigen DevOps-Auslieferungszyklus, in dem Designtime und Runtime sauber verknüpft sind und sich eine eindeutige Ende-zu-Ende-Verantwortung ergibt. Die Offenheit des Systems gestattet zudem die Nutzung der zahlreich vorhandenen Open-Source DevOps-Automatisierungs-Tools, wodurch das erhöhte Beschleunigungspotential bis hin zum Continuous Deployment kostengünstig ausgeschöpft werden kann. Darüber hinaus bietet SAP HANA auch eine hohe Flexibilität bezüglich der Nutzung verschiedener Programmiersprachen, da in der XSA individuelle Runtime-Umgebungen bspw. für Python eingesetzt werden können.

Im Hinblick auf die Anforderungen des zukunftsfähigen Data Warehouse, die sich vor allem in einem hohen Maß an Flexibilität und Agilität manifestieren, bietet diese SAP-Lösung speziell durch das konsequente Einhalten der DevOps-Methodik eine vielversprechende Ausgangsbasis.

FAZIT

In diesem Whitepaper haben wir die DevOps-Philosphie mit ihren Zielen, Charakteristika, Herausforderungen und Vorteilen präsentiert. Durch die im Nachgang aufgeführten Anforderungen des zukunftsfähigen Data Warehouse wurde deutlich, dass DevOps ein wirkungsvolles prozessuales Hilfsmittel zur Erfüllung der neuen Ansprüche gerade in punkto Flexibilität und Agilität darstellt.

Der innovative Ansatz des SAP SQL Data Warehousing macht die vollständige Adaption von DevOps möglich und baut auf ein dezidiertes Entwicklungsmodell auf Basis der klassischen DevOps-Phasen, Prozesse und Tools auf. Mit diesem Entwicklungsmodell und in Verbindung mit seiner modularen und offenen Struktur bietet das SAP SQL Data Warehousing beste Voraussetzungen, um den künftigen Herausforderungen des Datenmanagements zu begegnen.

Dabei werden viele Themen der klassischen Softwareentwicklung mitberücksichtigt.
Einige dieser Punkte sind:

- **Flexibilität**
  des Entwicklungsteams bei der Arbeit
- **Auslieferungeschwindigkeit**
  des Entwicklungsteams
- **Automatisierung**
  von wiederkehrenden Tätigkeiten
- **Auslieferungen**,
  die in sich konsistent sind und einen konkreten Mehrwert liefern
- **Offenheit**
  für neuartige Entwicklungswerkzeuge und offene Standards
- **Versionierung**
  einzelner Entwicklungsstände
- **Integration**
  des Entwicklungsprozesses in bestehende Unternehmensstrukturen

Insgesamt bleibt festzuhalten, dass auch die Data-Warehouse-Entwicklung sich Richtung DevOps öffnen wird und Anwender erkennen müssen, wie sie entsprechende Ansätze bestmöglich einbinden können. Das SAP SQL Data Warehousing offeriert hier eine ausgereifte Alternative, nicht nur für Nutzer anderer SAP-Software.

ÜBER ISR

Im Jahr 1993 gegründet, avancierte die ISR Information Products AG als Beratungsunternehmen mit Standorten in Braunschweig, Münster, Hamburg, Köln, München und Frankfurt zum Experten für Analytics, Prozess-Digitalisierung und Application Management.

Mit einem umfassenden Blick auf die Bedürfnisse namhafter Kunden konzipieren, modernisieren, implementieren und betreuen wir IT-Architekturen, Software-Lösungen und IT-Infrastrukturen.

Unsere ca. 200 Mitarbeiter schaffen dabei neue Lösungen im Umfeld SAP HANA, Big Data, Data Management, Cloud Computing und Künstliche Intelligenz (KI) für die Herausforderungen der digitalen Welt.

Unser Ziel:

Ihnen die wirtschaftliche Nutzung von Daten zu ermöglichen. Gezielte Analysen, strategisches Vorgehen, saubere Projektsteuerung und ganzheitliche IT-Lösungen sind für uns dabei die Basis, um Sie mithilfe verschiedener Digitalisierungsmaßnahmen fit zu machen für morgen.

for your digital smile

KONTAKTIEREN SIE UNS!

Ihr Ansprechpartner

Stefan Kahle

Senior Executive Manager | SAP Information Management
stefan.kahle@isr.de

ISR INFORMATION PRODUCTS AG

Hintern Brüdern 23
38100 Braunschweig
Tel. +49 (0) 531 12 08-0
hello@isr.de

Vorstand: Bernd Rosemeyer (Sprecher) | Manfred Merßmann
Aufsichtsratsvorsitzender: Peter Kallien
Sitz der Gesellschaft: Braunschweig