

Studienskript



SOFTWARE ENGINEERING FÜR DATENWISSENSCHAFTEN

DLMDWSEDW01

SOFTWARE ENGINEERING FÜR

DATENWISSENSCHAFTEN

IMPRESSIONUM

Herausgeber:
IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Postanschrift:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf
media@iu.org
www.iu.de

DLMDWSEDW01
Versionsnr.: 001-2023-1006
N.N.

© 2023 IU Internationale Hochschule GmbH
Dieses Lernskript ist urheberrechtlich geschützt. Alle Rechte vorbehalten.
Dieses Lernskript darf in jeglicher Form ohne vorherige schriftliche Genehmigung der
IU Internationale Hochschule GmbH (im Folgenden „IU“) nicht reproduziert und/oder
unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet wer-
den.
Die Autor:innen/Herausgeber:innen haben sich nach bestem Wissen und Gewissen
bemüht, die Urheber:innen und Quellen der verwendeten Abbildungen zu bestimmen.
Sollte es dennoch zu irrtümlichen Angaben gekommen sein, bitten wir um eine dement-
sprechende Nachricht.

INHALTSVERZEICHNIS

SOFTWARE ENGINEERING FÜR DATENWISSENSCHAFTEN

Einleitung

Wegweiser durch das Studienskript	6
Literaturempfehlungen	7
Pflichtliteratur	9
Übergeordnete Lernziele	10

Lektion 1

Agiles Projektmanagement	11
1.1 Klassisches Projektmanagement	12
1.2 Agiles Projektmanagement	19
1.3 Kanban	23
1.4 Scrum	28
1.5 Andere moderne Methoden	33
1.6 Einführung von Agile	37

Lektion 2

DevOps	41
2.1 Klassisches Lifecycle Management	43
2.2 Dev und Ops im Betriebsalltag	44
2.3 Auswirkungen auf die Team- und Entwicklungsstruktur	52
2.4 Aufbau einer DevOps-Infrastruktur	56
2.5 Skalierbare Umgebungen	65

Lektion 3

Softwareentwicklung	71
3.1 Testparadigmen und Überwachung	72
3.2 Entwicklungs- und Testansätze	78
3.3 Kontinuierliche Integration und kontinuierliche Lieferung	82
3.4 Versionsverwaltung	88
3.5 Entwicklertools	95

Lektion 4

API	103
4.1 Interaktion mit Software und Diensten	105
4.2 Designprinzipien für APIs	117
4.3 Erstellen einer Python-Bibliothek	124

Lektion 5	
Vom Modell zur Produktion	131
5.1 Modellentwicklungszyklus	132
5.2 Modellproduktionszyklus	140
5.3 MLOps und DataOps	148
5.4 Cloudbasierte Lösungen	157
Verzeichnisse	
Literaturverzeichnis	166
Abbildungsverzeichnis	180

EINLEITUNG

HERZLICH WILLKOMMEN

WEGWEISER DURCH DAS STUDIENSKRIPT

Dieses Studienskript bildet die Grundlage Deines Kurses. Ergänzend zum Studienskript stehen Dir weitere Medien aus unserer Online-Bibliothek sowie Videos zur Verfügung, mit deren Hilfe Du Dir Deinen individuellen Lern-Mix zusammenstellen kannst. Auf diese Weise kannst Du Dir den Stoff in Deinem eigenen Tempo aneignen und dabei auf lerntypspezifische Anforderungen Rücksicht nehmen.

Die Inhalte sind nach didaktischen Kriterien in Lektionen aufgeteilt, wobei jede Lektion aus mehreren Lernzyklen besteht. Jeder Lernzyklus enthält jeweils nur einen neuen inhaltlichen Schwerpunkt. So kannst Du neuen Lernstoff schnell und effektiv zu Deinem bereits vorhandenen Wissen hinzufügen.

In der IU Learn App befinden sich am Ende eines jeden Lernzyklus die Interactive Quizzes. Mithilfe dieser Fragen kannst Du eigenständig und ohne jeden Druck überprüfen, ob Du die neuen Inhalte schon verinnerlicht hast.

Sobald Du eine Lektion komplett bearbeitet hast, kannst Du Dein Wissen auf der Lernplattform unter Beweis stellen. Über automatisch auswertbare Fragen erhältst Du ein direktes Feedback zu Deinen Lernfortschritten. Die Wissenskontrolle gilt als bestanden, wenn Du mindestens 80 % der Fragen richtig beantwortet hast. Sollte das einmal nicht auf Anhieb klappen, kannst Du die Tests beliebig oft wiederholen.

Wenn Du die Wissenskontrolle für sämtliche Lektionen gemeistert hast, führe bitte die abschließende Evaluierung des Kurses durch.

Die IU Internationale Hochschule ist bestrebt, in ihren Skripten eine gendersensible und inklusive Sprache zu verwenden. Wir möchten jedoch hervorheben, dass auch in den Skripten, in denen das generische Maskulinum verwendet wird, immer Frauen und Männer, Inter- und Trans-Personen gemeint sind sowie auch jene, die sich keinem Geschlecht zuordnen wollen oder können.

LITERATUREMPFEHLUNGEN

ALLGEMEIN

Farcic, V. (2016). *The DevOps 2.0 toolkit: Automating the continuous deployment pipeline with containerized microservices*. CreateSpace Independent Publishing Platform.

Hunt, A., & Thomas, D. (1999). *The pragmatic programmer: From journeyman to master*. Addison-Wesley.

Kelleher, A. & Kelleher, A. (2019). *Machine learning in production: Developing and optimizing data science workflows and applications*. Addison-Wesley.

Kerzner, H. (2017). *Project Management - A Systems Approach to Planning, Scheduling, and Controlling* (12th ed., pp. 74–75). John Wiley & Sons.

Martin, R. C. (2008). *Clean code*. Prentice Hall.

LEKTION 1

Ries, E. (2011). *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Crown Publishing Group.

Cesarotti, V., Gubinelli, S., & Introna, V. (2019). The evolution of project management (PM): How agile, lean and six sigma are changing PM. *Journal of Modern Project Management*, 7(3), 1–29.

LEKTION 2

Kane, S. P., & Matthias, K. (2018). *Docker: Up and running*. O'Reilly Media.

Burns, B., & Beda, J. (2019). *Kubernetes: Up and running*. O'Reilly Media.

Huang, D., & Wu, H. (2017). *Mobile cloud computing: Foundations and service models*. Elsevier Science & Technology.

LEKTION 3

Kochhar, P. S., Xia, X., & Lo, D. (2019). Practitioners' views on good software testing practices. *Proceedings of the 2019 IEEE/ACM 41st international conference on software engineering: Software engineering in practice (ICSE-SEIP)*, 61–70.

Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*.

LEKTION 4

Clarke, S. (2004, May 1). *Measuring API usability*. Doctor Dobb's M-Dev.

Available online.

Spolsky, J. (2002, November 11). The law of leaky abstractions. *Joel on Software*. Available online.

Lane, K. (2019, October 10). Intro to APIs: History of APIs. *Postman*. Available online.

LEKTION 5

Ribeiro, J. L., Figueiredo, M., Araujo, A., Cacho, N., & Lopes, F. (2019). A microservice based architecture topology for machine learning deployment. *Proceedings of the 2019 IEEE international smart cities conference (ISC2)*, 426–431.

PFLICHTLITERATUR

LEKTION 1

Kerzner, H. (2017). *Project management - A systems approach to planning, scheduling, and controlling* (12th ed., pp. 287–288). John Wiley & Sons.

LEKTION 2

Johann, S. (2017). Kief Morris on infrastructure as code. *IEEE Software*, 34(1), 117–120.

Kneuper, R (2018). *Software processes and life cycle models: An introduction to modelling, using and managing agile, plan-driven and hybrid processes* (pp. 313–323). Springer.

Rossberg, J. (2014). Collaboration. In J. Rossberg, J. Ehn, & M. Olausson (Eds.), *Beginning application lifecycle management* (pp. 135–144). Apress.

LEKTION 3

Mahfuz, A. S. (2016). *Software quality assurance: Integrating testing, security, and audit*. CRC Press. Chapters 1–3

LEKTION 4

Lindman, J., Horkoff, J., Hammouda, I., & Knauss, E. (2020). Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. *IEEE Software*, 37(2), 52–59.

LEKTION 5

Lindman, J., Horkoff, J., Hammouda, I., & Knauss, E. (2020). Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. *IEEE Software*, 37(2), 52–59.

ÜBERGEORDNETE LERNZIELE

Der Kurs **Software Engineering für Datenwissenschaften** gibt Fachpersonen in datenintensiven Wissenschaften, z. B. in Data Sciences, einen umfassenden Überblick über die Grundlagen der Softwareentwicklung. Die erste Lektion führt zwei grundlegende Ansätze des Projektmanagements ein; die klassische und die agile Methode. Beide Ansätze werden heutzutage in Firmen praktiziert und bringen Vor- und Nachteile mit sich. Ganz gleich, welcher Ansatz am Arbeitsplatz gewählt wird, so bildet Projektmanagement die Grundlage produktiver Teamarbeit. Die Kenntnis bewährter Praktiken ist dabei hilfreich. Die zweite Lektion behandelt DevOps, einen modernen Ansatz, der die Softwareentwicklung und -produktion innerhalb einer Firma kombiniert. Die DevOps-Kultur stellt die Grundlage für die Bereitstellung und Pflege von Dienstleistungen in führenden Unternehmen wie Google dar, wird aber auch in vielen Startups angewandt. Die zweite Lektion bietet eine Einführung in die fundamentalen Bausteine skalierbarer Softwareumgebungen nach der DevOps-Philosophie. Das Wesentliche der Softwareentwicklung wird in der dritten Lektion behandelt. Diese beginnt mit dem Testen von selbstentwickelter Software und der Automatisierung von Tests in der Produktionspipeline. Des Weiteren werden die Versionsverwaltung und die von Entwicklerinnen und Entwicklern in der Praxis verwendeten Werkzeuge abgedeckt. Die Anwendungsprogrammierschnittstellen, auch APIs genannt, bieten Interaktionsmöglichkeiten mit der Software und deren Diensten. Darum geht es in der vierten Lektion. Ferner werden die wichtigsten Prinzipien zum Design und zur Erstellung von guten Benutzeroberflächen erklärt, die Sie daraufhin anwenden können, um eine kleine Bibliothek in Python zu erstellen. In der letzten Lektion lernen Sie Unterschiede im Arbeitsablauf von Fachkräften in den Datenwissenschaften und in der Softwareentwicklung kennen. Es wird darauf eingegangen, wie sich deren Anforderungen und Erwartungen unterscheiden und wie die Arbeit von Datenwissenschaftlerinnen und -wissenschaftlern sicher in skalierbare Produktionsumgebungen einfließen kann.

LEKTION 1

AGILES PROJEKTMANAGEMENT

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- klassisches Projektmanagement zu erkennen; welche Arten es gibt, und wie es definiert wird.
- agiles Projektmanagement zu erkennen und zu definieren.
- Kanban- und Scrum-Methoden zu erkennen und zu definieren.
- wiederzugeben, welche modernen Methoden im Projektmanagement gebraucht werden.
- einen Übergang vom klassischen zum agilen Projektmanagement zu gestalten.

1. AGILES PROJEKTMANAGEMENT

Einführung

Die Arbeit einer Abteilung oder eines Teams lässt sich in zwei Bereiche unterteilen: den laufenden Betrieb und die neuen Projekte. Im laufenden Betrieb werden tägliche Vorgänge abgedeckt. In einer Softwareentwicklungsabteilung eines Autoherstellers besteht das Hauptziel beispielsweise in der Programmierung der für das Unternehmen notwendigen Software. Die meist von der Betriebsabteilung ausgeführten Aufgaben sind nicht an bestimmte Termine gebunden und sollten zur Lösung von Problemen und zu Ergebnissen durch wiederholbare Vorgänge und Aktivitäten führen. In der Projektabteilung werden dagegen Projekte, also spezielle, durch einmalige Ergebnisse und eindeutige Anfangs- und Endzeiten definierte Aufgaben durchgeführt. Die Projekte verkörpern keine langfristigen Ziele einer Abteilung, da sie generell temporär und von kurzer Dauer sind. Aus den Projekten geht jedoch für ein Unternehmen überlebenswichtige Innovation hervor. Mit steigender Wichtigkeit von Innovationen werden auch Projekte und damit das Projektmanagement und die entsprechenden Fähigkeiten immer bedeutsamer.

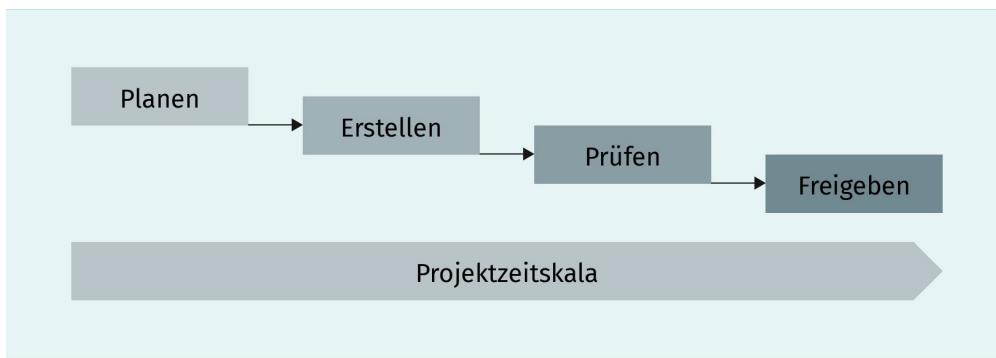
Die Softwareentwicklungsabteilung eines Autoherstellers könnte zum Beispiel mit einem Verkaufs- und Marketingprojekt beauftragt werden oder mit einem Entwicklungsprojekt für ein Autonavigationssystem, sodass das Unternehmen auf dem Markt konkurrenzfähig bleibt. Dazu braucht ein Unternehmen Projektmanagement, damit ein Team alle Projekte ausführen, erfolgreich zum Abschluss bringen und qualitativ hochwertige Ergebnisse erzielen kann. Das Projektmanagement beinhaltet Werkzeuge, Theorien, Techniken, Wissen und Fähigkeiten, die zur Teamführung, dem Ressourceneinsatz und dem Management von komplexen Projekten notwendig sind. Es sollen vordefinierte Ziele mit bestimmten Erfolgskriterien erreicht und gleichzeitig ein bestimmter Zeitrahmen und ein Budget eingehalten werden. In den letzten Jahrzehnten haben verschiedene Managementorganisationen versucht, Leitlinien einzuführen, die alle Aspekte des Projektmanagements abdecken. In den Fünfzigerjahren des vergangenen Jahrhunderts begann die Industrie, maßgeblich zu wachsen und Unternehmen setzten systematisch Werkzeuge und Methoden des Projektmanagements in den Entwicklungsphasen ihrer Projekte ein (Kwak, 2005). Eine Leitlinie, die bis ins Jahr 2000 sehr intensiv zum Einsatz kam und die auch heute noch gebraucht wird, ist das klassische Projektmanagement (Project Management Institute, 2017).

1.1 Klassisches Projektmanagement

Das klassische Projektmanagement besteht aus Techniken und Werkzeugen, die auf Aktivitäten oder Aufgaben angewandt werden können, die zur Erstellung eines Endprodukts, für ein bestimmtes Ziel oder eine bestimmte Dienstleistung gedacht sind. Diese Leitlinie gründet auf dem in der nachstehenden Abbildung dargestellten **Wasserfallmodell**:

Wasserfallmodell
Eine lineare sequenzielle Entwicklungsmethode, bei der jede Phase bzw.

Abbildung 1: Wasserfall-Workflow

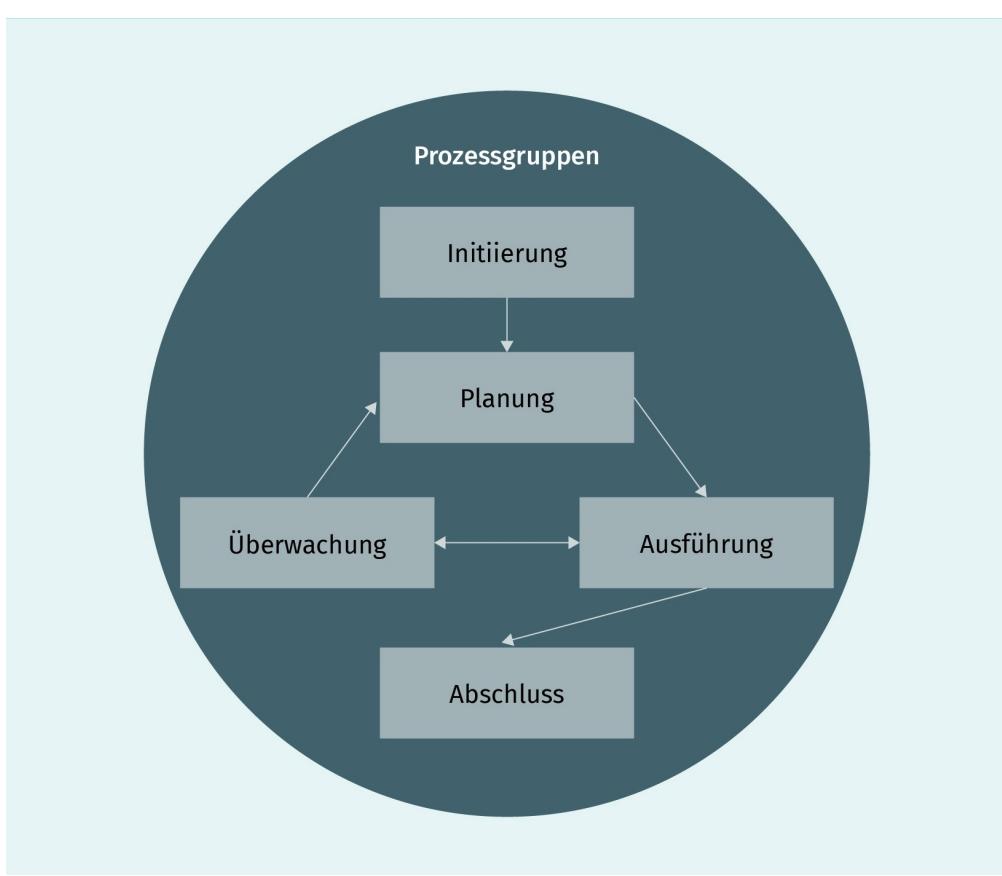


jeder Schritt erst dann beginnt, wenn der vorherige Schritt (z.B. Softwaretesten) abgeschlossen ist.

Quelle: Asadi (2020).

Die Hauptidee des klassischen Projektmanagements ist die planbare und lineare Umsetzung von Projekten nach klar definierten und abgrenzbaren Schritten, sodass ein Plan nicht grundlegend verändert werden muss (Project Management Institute, 2017). In der heutigen Zeit verfügen manche Projekte über klare Ziele und müssen während ihrer Laufzeit nicht oft angepasst werden. Somit können Ziele durch lineare, sequenzielle Ereigniskreisläufe oder -phasen erreicht werden. Für derlei Projekte eignet sich das klassische Projektmanagement, da dieser Ansatz einen festen Lebenszyklus mit den folgenden Schritten beinhaltet.

Abbildung 2: TPM-Workflow



Quelle: Department of Veteran Affairs (2020).

Nachdem eine Idee entsteht, wird innerhalb des Projekts ein neues Projekt oder eine neue Phase erstellt. Das Managementteam beginnt damit, die notwendigen Ressourcen, den Projektumfang, die Projektziele und deren Umsetzung zu bestimmen. Anschließend beginnt das Entwicklungsteam mit der Ausführung der im Projektplan definierten Aufgaben, während Status und Fortschritt jeder Phase vom Management überwacht werden. Nachdem alle Aktivitäten abgeschlossen sind und das Ziel erreicht wurde, schließt der:die Manager:in das Projekt bzw. eine Phase offiziell ab. Beim klassischen Projektmanagement ist das Projekt meist in klar definierte Phasen unterteilt, die jeweils eigene Zielvorgaben und Ergebnisse haben. Bevor von einer Phase in die nächste übergegangen wird, müssen das Ziel der vorherigen erreicht, die Aufgaben erfüllt und alle Entscheidungen getroffen sein. Im klassischen Projektmanagement wird besonderer Wert auf lineare Phasen, Vorausplanung, Dokumentation und Priorisierung der definierten Aufgaben gelegt. Im gesamten Projektlauf wird durch klassisches Projektmanagement gewährleistet, dass die vordefinierten Phasen befolgt werden. Beim Entwurf des Gesamtplanes, der alle Anforderungen und Risiken jeder Phase abdeckt, geht man beim klassischen Projektmanagement davon aus, dass sich die Umgebung und die Ressourcen während des gesamten Projekts nicht ändern. Das Ergebnis wird dabei in kleinere Aufgaben unterteilt, wodurch diese zunehmend berechenbarer werden. Dank gründlicher Prognosen bzw. Projektionen dieser Aufgaben und der damit verbundenen Risiken kann beim klassischen Projektmanagement

ein Projekt kontrolliert durchgeführt werden. Das Ziel dieser Strategie besteht darin, die Effizienz des ursprünglichen Projektplans zu steigern und dadurch die Ergebnisse im entsprechenden Zeitplan, Budget und Projektumfang zu erreichen.

Das klassische Projektmanagement hat die folgenden Vorteile:

- Aufgrund der sequenziellen und kontrollierbaren Prozesse ist es leicht verständlich und ausführbar.
- Durch feste Annahmen und die daraus resultierende Verringerung der Projektkosten ist das klassische Projektmanagement kostengünstig.
- Es ist effizient, da standortunabhängige Projekte in der Regel weniger Kommunikation, aber einen genaueren Plan erfordern. Für beide Aspekte erweist sich das klassische Projektmanagement als geeignete Methode.
- Das Erreichen von Zielvorgaben innerhalb geplanter Zeiträume führt zu einer erhöhten Kundenzufriedenheit.

Allerdings gibt es auch folgende Nachteile:

- Da der Ansatz auf Prognosen beruht, kann es zu Verzögerungen im Ablauf kommen, wenn tatsächliche Kosten, Aufwände und Ressourcen von der ursprünglichen Schätzung abweichen.
- Im Vergleich zu anderen Methoden, die in dieser Lektion behandelt werden, findet weniger Austausch mit den Kund:innen statt. Das kann zu Missverständnissen bezüglich des Ziels oder einzelner Ergebnisse des Projekts führen.
- Auch die Kreativität von Mitarbeitenden kann durch die vorgegebenen und sequenziellen Phasen zu kurz kommen.
- Wenn sich ein einzelnes Teammitglied nur um eine Aufgabe und den damit verbundenen persönlichen Einsatz, das geforderte Wissen und das Ergebnis kümmert, kann es zu Koordinationsproblemen kommen.

Verschiedene Methoden führen detaillierte Beschreibungen jedes Schritts bzw. jeder Phase sowie Werkzeuge und Vorgehensweisen ein. PMBOK und PRINCE2 gehören zu den bekannten Methoden des klassischen Projektmanagements, die in dieser Lektion dargestellt werden.

PMBOK

Project Management Body of Knowledge, kurz PMBOK, wurde durch das Project Management Institute (PMI) eingeführt. Das im Jahre 1969 gegründete PMI ist eine Non-Profit-Organisation, die Unterstützungsangebote zur Verfügung stellt und weltweite Standards für das Projektmanagement entwickelt (Project Management Institute, 2017). Das PMBOK wurde 1983 zum ersten Mal als eine Sammlung von detaillierten Richtlinien und genormter und definierter Projektmanagementterminologie veröffentlicht. Es wird von Manager:innen zur Standardisierung von Projekten in verschiedenen Teams und Organisationen gemäß PMI verwendet. Dem PMBOK liegt die Annahme zugrunde, dass Projektmanager:innen einen Standard benötigen, der sich kulturunabhängig auf Projekte jeglicher Art, in allen Industriebereichen und -sektoren anwenden lässt. Es umfasst fünf Grundprozesse und zehn Wissensbereiche. Diese Wissensbereiche hängen zwar zusam-

men, sind aber nicht alle auf jeden einzelnen Prozess anwendbar. Bei den fünf, dem klassischen Projektmanagement entstammenden Prozessen, die in fast allen Projekten vorkommen, handelt es sich um Initiierung, Planung, Ausführung, Überwachung und Abschluss. Jeder Prozess beschreibt die Aufgaben, die zur Durchführung aller Projekte notwendig sind. Sie werden als Input (Dokumente, Pläne, Design, usw.), erwartetes Output (Dokumente, Pläne, Design, usw.) und Werkzeuge oder Verfahrensweisen, durch die Inputs zu den entsprechenden Outputs werden, definiert.

Das Output eines Prozesses stellt den Input für den nächsten dar. Jeder Prozess sollte ein bestimmtes Erfordernis des Projekts abdecken. Das Managementteam muss festlegen, welcher Prozess von wem, wie und wann verwendet wird.

Das PMBOK listet bewährte Methoden, Konventionen und Verfahrensweisen auf, die Industrienormen entsprechen und Teams oder Organisationen bei der Ausführung von Projekten unterstützen. Die Best Practices im PMBOK sind eine Art generelles Einvernehmen, nach dem die Anwendung von Wissen, Werkzeugen und Verfahrensweisen des PMBOK die Erfolgschance der verschiedensten Projekte erhöhen kann (Project Management Institute, 2017). Das PMI versucht, die mit dem PMBOK eingeführten Richtlinien regelmäßig zu aktualisieren und die jeweils neuesten Projektmanagementpraktiken zugrunde zu legen und dabei auch die wachsende Bedeutung von Neuerungen einzubeziehen. Die derzeitige Version des PMBOK wurde 2017 herausgegeben (Project Management Institute, 2017). Darin werden zehn verschiedene Wissensbereiche behandelt, durch die festgelegt ist, welche Prozesse für effektives Projektmanagement ausgeführt werden müssen. Diese Wissensbereiche werden nachfolgend aufgeführt:

1. Durch Integrationsmanagement werden verschiedene Prozesse definiert und innerhalb der unterschiedlichen Prozessgruppen zusammengeführt.
2. Das Umfangsmanagement legt die Arbeit zur erfolgreichen Prozessausführung fest.
3. Die Terminplanung gewährleistet den zeitgerechten Abschluss des Projekts.
4. Das Kostenmanagement umfasst die Kostenkontrolle und -schätzung sowie die Budgetierung des Projekts, sodass ein Projekt im Rahmen des genehmigten Budgets abgeschlossen werden kann.
5. Im Qualitätsmanagement werden Richtlinien festgelegt, mithilfe derer hochwertige Endergebnisse erzielt werden können.
6. Das Personalmanagement bedeutet hier die Organisation und die Führung von Projektteammitgliedern und das Management weiterer Betriebsmittel.
7. Zur Kommunikation gehören die Erstellung, Sammlung, Verteilung, Verwaltung und Überwachung von Projektinformationen.
8. Das Risikomanagement deckt die Identifizierung, Planung, Analyse und die Überwachung von in jeder Phase des Projekts möglichen Risiken ab.
9. Zur Beschaffung gehört die Bestimmung und der Erwerb von externen Dienstleistungen und Produkten, die zur Projektausführung notwendig sind.
10. Beim Stakeholder:innenmanagement müssen Personen bzw. Organisationen, die Einfluss auf das Projekt haben könnten, identifiziert und es muss an sie herangetreten werden.

Das PMBOK ist keine Methode, sondern ein Leitfaden mit Wissen, das den Lebenszyklus eines Projekts oder eines Programms bestimmt. In jeder Branche bzw. Organisation können je nach Bedarf die Best Practices nach den im PMBOK ausgeführten Prozessen erstellt werden. Ein Nachteil dieser Methode könnte darin liegen, dass die zahlreichen Kombinationen an Prozessen und Wissensbereichen für Projekte kleineren Umfangs möglicherweise zu kompliziert sind; daher müssen Umfang und Größe der Anwendung auf jedes Projekt angepasst werden.

PRINCE2

PRINCE2 steht für „Projects in Controlled Environments“ bzw. „Projekte in kontrollierten Umgebungen“ und wurde im Jahr 1990 von der britischen Regierung zur Unterstützung des Projektmanagements in der IT-Branche, aber auch in anderen Industriezweigen eingeführt und veröffentlicht. Im Lauf der Zeit entwickelte es sich zu einem bekannten Standard und wurde von Institutionen der Europäischen Union angewandt (Siegelaub, 2004). Im Gegensatz zum PMBOK, einer Sammlung von Richtlinien bzw. Standards, ist PRINCE2 ein generisches Projektmanagement-Rahmenwerk, das als branchenunabhängiger Standard mit Schwerpunkt sowohl auf dem Prozess als auch einem hochwertigen Endprodukt entwickelt wurde. Daher ist PRINCE2 als ein perspektiven- und produktbasiertes Projektmanagement-Modell aufzufassen. Der inhärente Produktfokus von PRINCE2 spiegelt die Realität und das Projektziel „Geschäftsbedarf als Projektergebnis“ wider. PRINCE2 stellt eine eigenständige Methode dar, deren Rahmenwerk auf jeweils sieben Prinzipien, Themen und Prozessen basiert, die Projektmanager:innen anwenden und auf die Anforderungen jedes Projekts anpassen müssen (Siegelaub, 2004). Die folgenden Prinzipien gelten als die Grundprinzipien von PRINCE2:

1. Fortlaufende geschäftliche Rechtfertigung
2. Lernen aus Erfahrung
3. Definierte Rollen und Verantwortlichkeiten
4. Steuerung über Managementphasen
5. Managen nach dem Ausnahmeprinzip
6. Fokus auf Produkten
7. Anpassen an die Projektsituation

Im Lebenszyklus spielen auch sieben Themen bei PRINCE2 eine Rolle. Sie müssen ständig in die Projektaufgaben und -aktivitäten einfließen:

1. Business Case – Dieses Thema beruht auf dem Prinzip der fortlaufenden geschäftlichen Rechtfertigung, anhand dessen bewertet wird, ob ein Projekt rentabel und realisierbar ist.
2. Änderung – Durch dieses Thema wird eruiert, wie mit während des Projekts notwendigem Änderungsbedarf umgegangen wird.
3. Organisation – Im Rahmen dieses Themas werden Rollen und Verantwortlichkeiten festgehalten.
4. Pläne – Durch Pläne wird beschrieben, wie das Team sein Ziel eines Produkts innerhalb des festgelegten Budgets und Zeitrahmens erreicht.
5. Fortschritt – Hierbei wird der Fortschritt festgehalten, sodass das Team einen ständigen Überblick über den aktuellen Status hat.

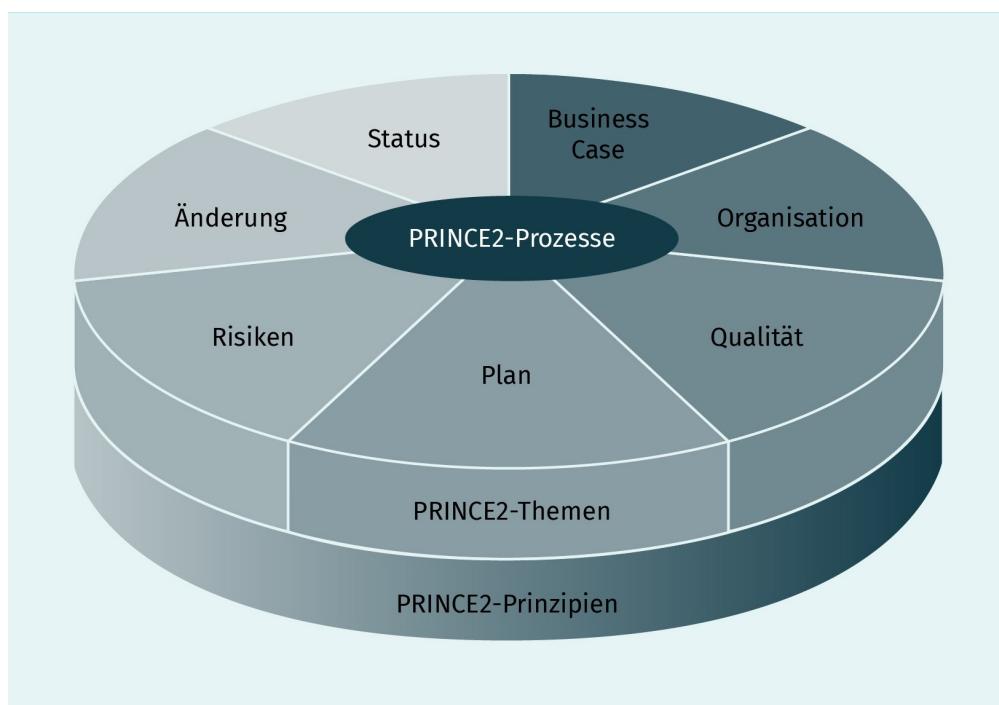
6. Qualität – Dieses Thema bezieht sich auf das Prinzip „Fokus auf Produkten“ und gewährleistet die Qualitätskontrolle in jeder Projektphase.
7. Risiko – Dadurch werden die im Projekt möglichen Risiken identifiziert und kontrolliert.

Diese Themen entsprechen in etwa den Wissensbereichen des PMBOK. Zu PRINCE2 gehörten weiterhin sieben Prozesse, die zur Realisierung der Projektziele der Reihe nach implementiert werden sollten. Zu diesen sieben Prozessen gehören:

1. das Vorbereiten eines Projekts,
2. die Initiierung eines Projekts,
3. das Lenken eines Projekts,
4. die Steuerung einer Phase,
5. das Managen der Produktlieferung,
6. das Managen von Phasenübergängen, und
7. das Abschließen eines Projekts.

Die nachstehende Graphik zeigt die Struktur von PRINCE2.

Abbildung 3: PRINCE2-Struktur



Quelle: Ranjan (2014).

Da es innerhalb der sieben Prozesse von PRINCE2 verschiedene Aktivitäten gibt, müssen jeweils die Verantwortlichen und der Zeitpunkt ihrer Ausführung festgelegt werden (Bentley, 2010). Dank der klaren Definition des Vorgehens von PRINCE2 ist das Projektmanagement auch für Projektmanager:innen mit wenig Erfahrung ein leichtes Unterfangen. Außerdem wird dadurch die Qualität des Projektergebnisses verbessert. Trotz des gut defi-

nierten Workflows ist die Anwendung von PRINCE2 keine Erfolgsgarantie für Projekte mit komplexerem Projektmanagementbedarf. Darüber hinaus kann PRINCE2 für kleinere Projekte der falsche Ansatz sein, da der Fokus bei PRINCE2 auf einer so großen Anzahl von verschiedenen Aktivitäten liegt.

1.2 Agiles Projektmanagement

Obwohl in den letzten Jahren viele effiziente Projektmanagement-Methoden eingeführt wurden, gibt es keine Universalmethode, die zum Management und zur erfolgreichen Durchführung aller Projekte verwendet werden kann. Das klassische Projektmanagement eignet sich für Projekte mit geringem Unsicherheitsgrad, klaren Anforderungen mit wenigen Änderungen, klaren Zielen und wenig Kund:innenauftausch. Das Hauptaugenmerk der auf dem klassischen Projektmanagement beruhenden Methoden liegt in der Ausgangsplanung und der Prognose der einzelnen Projektphasen. Während des Projektablaufs ist es beim klassischen Projektmanagement nicht notwendig, die Kund:innen oder die Endverbraucher:innen einzubeziehen, da der Schwerpunkt auf der Dokumentation jeder Aktivität liegt. Für Projekte, die sich nicht für das klassische Projektmanagement eignen, könnte das agile Projektmanagement, kurz **Agile** genannt, eine Alternative bieten. Diese Methode entstammt der Softwareentwicklung und wurde durch ein kurzes, 2001 publiziertes Dokument, dem Manifest für Agile Softwareentwicklung, in dem vier zentrale Werte und zwölf Prinzipien dargelegt wurden, eingeführt (Fowler & Highsmith, 2001). Die vier im Agilen Manifest festgelegten Werte lauten:

1. **Personen und Interaktionen stehen über Prozessen und Werkzeugen:** Kommunikation und Zusammenarbeit mit Kund:innen und anderen ist wichtiger als standardisierte Prozesse und Werkzeuge. Zentral ist auch die effektive Zusammenarbeit aller Projektbeteiligten und Teammitglieder.
2. **Funktionierende Software steht über einer umfassenden Dokumentation:** Beachtung wird der Lieferung funktionsfähiger Anwendungen und nicht so sehr einer umfassenden Dokumentation geschenkt. Das Ziel ist nicht die Erstellung von Dokumentation, sondern eines Produkts.
3. **Die Zusammenarbeit mit Kund:innen steht über der Vertragsverhandlung:** Die Unterzeichnung eines Vertrages ist weniger wichtig als die konkrete Zusammenarbeit. Ein Vertrag muss am Anfang sicherlich vorhanden sein. Mit der Zeit können sich Kund:innenanforderungen allerdings ändern, was unter Umständen im ursprünglichen Vertrag nicht festgehalten wird.
4. **Reagieren auf Veränderung steht über dem Befolgen eines Plans:** Da das Reagieren auf Änderungen wichtiger ist als die Ausführung des Plans, muss Flexibilität ein wichtiges Prozesselement sein, sodass der volle Projektumfang gewährleistet werden kann.

Agile
Ein iterativer Ansatz zur Lieferung eines Projekts durch iterative Entwicklungsschleifen, den sogenannten Sprints.

Wie bereits erwähnt, enthält das Manifest für Agile Softwareentwicklung zwölf Prinzipien für das agile Projektmanagement (Fowler & Highsmith, 2001). Diese Prinzipien beschreiben den flexiblen Ansatz von Agile in Bezug auf Änderungen und den Fokus auf Kund:innenkommunikation und werden nachfolgend erläutert.

1. **Kund:innenzufriedenheit:** Sie wird durch die frühe und kontinuierliche Lieferung eines Wertproduktes gewährleistet. Kund:innen sind zufriedener, wenn sie, anstatt auf die Herausgabe zu warten, regelmäßige Lieferungen funktionierender Software erhalten.
2. **Anforderungsänderungen:** Änderungen der Anforderungen werden fortlaufend erhalten und während des gesamten Entwicklungsprozesses implementiert.
3. **Lieferung:** Funktionsfähige Software oder Produkte sollten, wenn möglich wöchentlich statt monatlich, an die Kund:innen geliefert werden.
4. **Zusammenarbeit:** Fachpersonen und Entwickler:innen sollten während des Projekts eng zusammenarbeiten. Entscheidungen, über die sich Kund:innen bewusst sind und an denen sie teilhaben, sind in der Regel deutlich besser.
5. **Mitarbeitendenunterstützung:** Personen, die am Projekt mitarbeiten, sollten motiviert sein. Sie müssen unterstützt werden und ihnen muss Vertrauen zukommen.
6. **Kommunikation:** Nach Auswahl des Entwicklungsteams müssen die persönliche Kommunikation und die Zusammenarbeit unterstützt werden. Sie sind der Schlüssel zum Erfolg des Projekts.
7. **Fortschrittsmessung:** Der Hauptmaßstab des Fortschritts ist eine funktionierende und funktionelle Software.
8. **Nachhaltigkeit:** Agile Prozesse führen zu nachhaltiger Entwicklung; Kund:innen, Entwickler:innen sowie Stakeholder:innen sollte es möglich sein, das Produkt auf Dauer weiterzuentwickeln.
9. **Exzellenz:** Es muss stetig Wert auf technisch exzellente Ausführung, Detail und Design gelegt werden.
10. **Einfachheit:** Das Projekt sollte einen klaren Zweck haben und eindeutigen, allseits verständlichen Prinzipien folgen, sodass jede:r eine Änderung vorschlagen kann.
11. **Architektur:** Die beste Architektur, gute Anforderungen und ausgezeichnetes Design kommen von selbstorganisierten Teams. Wenn Menschen motiviert sind, übernehmen sie mehr Verantwortung und treffen bessere Entscheidungen auch dadurch, dass sie mit anderen kommunizieren. Das führt zu außerordentlicher Zusammenarbeit und einem hochwertigen Produkt.
12. **Feedback:** Es sollte immer wieder Feedback zur Effektivitätsverbesserung angeboten werden. Denn Teams werden durch persönliche Weiterbildung sowie verbesserte Fähigkeiten, Werkzeuge und Prozesse effizienter.

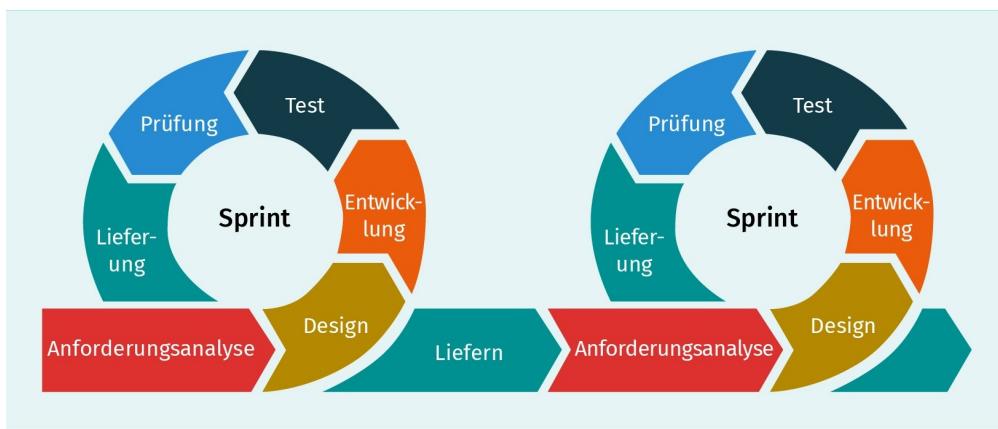
Nach Erläuterung der Werte und Prinzipien des agilen Projektmanagements können wir tiefer in diese Methode einsteigen. Agile ist eine iterative Form des Projektmanagements, die Organisationen dabei unterstützt, Produkte und Dienstleistungen schnell und problemlos an ihre Kunden zu liefern. Die Struktur dieser Methode entspricht den verschiedenen Ebenen einer Organisation, z. B. Personal, Teams, Abteilungen (Highsmith, 2009). Die Anwendung agiler Methoden auf die Projektentwicklung kann, je nachdem wie die Arbeit einer Organisation gestaltet ist, die Zusammenarbeit und Leistung des Teams fördern. Tatsächlich beruht Agile hauptsächlich auf Teamwork, Kommunikation, Zusammenarbeit, Timeboxing und dem zügigen Reagieren auf Änderungen, die im Laufe des Projektlebenszyklus anfallen. Ein besonderer Schwerpunkt liegt auch auf dem Software- und Projektentwicklungsprozess und auf der Qualität von Prozessen, der Software und der Projekte. Um Lieferungen zu gewährleisten und die Projektzeit im agilen Prozess zu verkürzen, müssen folgende Eigenschaften berücksichtigt werden:

- **Iterationen:** Beim agilen Projektmanagement steht während der gesamten Dauer eines Kurzyklus, der aus schneller Überprüfung, Korrektur und Ausführung (anhand mehrerer Iterationen) besteht, eine einzige Anforderung im Mittelpunkt.
- **Modularität:** Während der Entwicklungsphase werden bei Agile ganze Systeme in kleine, überschaubarere Module eingeteilt.
- **Timeboxing:** Zur Anwendung der agilen Methode steht einem Team, einem: einer Entwickler:in in bestimmter Zeitraum in Iterationszyklen von einer bis sechs Wochen zur Abarbeitung einer Aufgabe innerhalb eines Moduls zur Verfügung.
- **Optimierung:** Alle unnötigen Aktivitäten werden bei der agilen Methode außen vorgenommen, um Risiken zu vermeiden und das Ziel zu erreichen.
- **Anpassung:** Die Anpassung an neue Risiken oder Änderungen findet beim agilen Projektmanagement schnell statt.
- **Inkrementen:** Produkte müssen bei Agile in Inkrementen entwickelt werden, wodurch funktionelle Anwendungen in kleinen Schritten gebaut werden können. Zum Schluss werden die Inkremeante in ein komplettes System oder Produkt integriert.
- **Konvergenz:** Im agilen Prozess fließen die Risiken der Inkrementen zusammen.
- **Zusammenarbeit:** Der Agile Ansatz ist von Natur aus kommunikativ und kollaborativ.
- **Kund:innenorientierung:** Kund:innenzufriedenheit hat oberste Priorität, das heißt, dass Personen über den Prozessen oder der Technologie stehen.

Die agile Methodik basiert auf einem iterativen Prozess, in dem Projekte in kleine Teile oder Inkrementen, die als **Sprints** bezeichnet werden, unterteilt werden. Der Workflow bei Agile besteht aus Sprints, die dem Softwareentwicklungszyklus folgen und wozu das Verständnis von Anforderungen, Design, Entwicklung, Überprüfung und Bereitstellung oder Lieferung notwendig ist. Die folgende Abbildung zeigt ein Beispiel dafür:

Abbildung 4: Workflow im agilen Prozess

Sprint
Ein kurzer Zeitraum, während dessen im Rahmen eines Scrums ein bestimmtes Arbeitsvolumen abgearbeitet werden sollte.



Quelle: Asadi (2020).

Im Großen und Ganzen bedeutet Agile weniger Zeit für die Planung oder die Priorisierung und mehr Zeit dafür, die Kund:innenanforderungen zu verstehen und Ergebnisse zu liefern. Da die Kund:innenzufriedenheit bei Agile oberste Priorität hat, sind Kund:innen bei allen Iterationen involviert und werden dazu aufgefordert, das gelieferte Produkt am Ende einer jeden Phase bzw. eines jeden Sprints zu überprüfen. Da Iterationen inhärent sind, ist diese Methode flexibler und kann daher leichter an die zur Kund:innenzufriedenheit not-

wendigen Änderungen angepasst werden. Alle neuen Anforderungen werden normalerweise in der nächsten Schleife der Iteration berücksichtigt. Dadurch, dass Projekte in kleinere Abschnitte unterteilt werden, hat das Management mehr Kontrolle, wodurch das damit verbundene Risiko besser gemanagt und Probleme gleich bei Entstehen beseitigt werden können. Bei der agilen Methodik gibt es auch verschiedene Rollen, die unter den an der agilen Projektentwicklung beteiligten Teammitgliedern aufgeteilt werden müssen.

- **Benutzer:in oder Kund:in:** Der agile Prozess beginnt immer mit den Kund:innen und ihren Anforderungen.
- **Produktinhaber:in:** Der:die Produktinhaber:in (im Deutschen oft als Product Owner bezeichnet) verleiht den Kund:innen und internen Stakeholder:innen eine Stimme, analysiert Ideen und fasst sie zusammen, sammelt erforderliches Wissen und gibt Produktfeedback. Er:Sie arbeitet mit dem Entwicklungsteam, erstellt aus den Kund:innenanforderungen **Benutzergeschichten** und liefert zusätzliche Details. Diese sollten abdecken, wer der:die Benutzer:in bzw. Kund:in ist, welche Art von Problem für ihn:sie gelöst wurde, warum die neue Lösung wichtig ist und welche Akzeptanzkriterien auf die Lösung anwendbar sind.
Ein:e Produktinhaber:in ist ein Teammitglied, das die Verantwortung für die Formulierung der Vision und für die Erarbeitung einer daraus hervorgehenden, akzeptablen Lösung mit dem Entwicklungsteam trägt.
- **Produkt- oder Softwareentwicklungsteam:** Bei der agilen Methodik setzt ein hochqualifiziertes Team seine Fähigkeiten zur Projektfertigstellung und zur Lieferung eines funktionellen Produkts ein. Dazu finden häufig, manchmal sogar täglich, Meetings oder Absprachen statt, um den Fortschritt und die Verantwortlichkeiten für bestimmte Aufgaben zu überprüfen.

Vor- und Nachteile von Agile

Das oberste Ziel von Projektmanagement ist es, Ergebnisse termin- und budgetgerecht zu liefern. Wie andere Methoden hat auch Agile Vor- und Nachteile, die ein Team dabei beeinflussen, dieses Ziel zu erreichen. Diese Vor- und Nachteile müssen daher bei Einführung des agilen Prozesses in einer Organisation bekannt sein. Es sollten unter anderem die folgenden Vorteile für das Team und die Organisation beachtet werden, wenn die Eigenschaften und Prinzipien des agilen Projektmanagements abgewogen werden:

- Am Beginn des Projektes müssen die Anforderungen nicht so genau festgelegt werden, da während des Projekts weitere Informationen eingeholt werden können (Sharma et al., 2012).
- Gute, persönliche Kommunikation zwischen den Entwicklerteams und den Kund:innen sind die Grundlage von Agile. Dadurch kann auch das Entwicklungsrisiko reduziert werden (Sharma et al., 2012).
- Die vorzeitige Lieferung eines Produkts ist gut für Kund:innen (Masson et al., 2007).
- Produkte, die mit der agilen Methode entwickelt wurden, können schneller auf dem Markt eingeführt werden als Produkte, die anhand der klassischen Methode entwickelt wurden (Carilli, 2013).
- Die Projektkosten sind bei agilen Projekten generell niedriger als bei der klassischen Methode (Carilli, 2013).

Benutzergeschichte

Eine Benutzergeschichte ist eine bei der agilen Methodik verwendete, vereinfachte Beschreibung des Kunden / der Kundin und dessen / deren Lösungsbedarfs.

- Wenn Änderungen angenommen und Kund:innen bei jedem Schritt bzw. jeder Entscheidung miteinbezogen werden, steigt die Qualität des Endprodukts (Masson et al., 2007).

Agile ist dennoch kein perfektes Modell und kann den Erfolg eines Projekts nicht garantieren. Daher sollten auch die folgenden Nachteile in die Entscheidung über den Projektmanagementansatz einfließen:

- Da die Anfangsplanung bei Agile weniger detailliert ist, können daraus inkorrekte Aufwands- und Kostenschätzungen resultieren (Serrador & Pinto, 2015).
- Im agilen Prozess wird weniger dokumentiert, wodurch dem Entwicklungsteam, aber auch nachfolgenden Teams oder neuen Entwickler:innen eventuell weniger Informationen zur Verfügung stehen (Sharma et al., 2012).
- Die Verbesserung eines Produkts beruht beim agilen Prozess auf klarem Feedback von Kund:innen und guter Kommunikation. Wenn Kund:innen nicht repräsentativ sind oder kein klares Feedback geben können, wirkt sich das erheblich auf die Entwicklungsgeschwindigkeit des Teams aus (Sharma et al., 2012).
- Ein Team muss aus erfahrenen Entwickler:innen bestehen, die schnell arbeiten können und wenig Zeit zur Projektplanung benötigen. Dadurch entstehen höhere Kosten (Taibi et al., 2017).

Trotz dieser Nachteile ist Agile eine erfolgreiche Methode, die in der IT-Branche zur Entwicklung qualitativ hochwertiger Produkte geführt hat. Bekannte Firmen wie IBM, Microsoft, Apple und Adobe, verwenden heute agiles Projektmanagement, um Programme anhand von sogenannten agilen Transformationen zu ändern und zu aktualisieren (Carilli, 2013). Transformationen sind umfangreiche, organisatorische Änderungen, bei denen agile Methoden in kleinen, multidisziplinären Teams angewandt werden, um schnell, experimentell und iterativ Ergebnisse vorzulegen (Consultancy, 2020). Im Jahr 2012 hat beispielsweise Adobe Agile mit dem Ziel eingeführt, den Managementaufwand zu reduzieren und konnte dadurch jährlich 80.000 Arbeitsstunden einsparen (Hearn, 2019). Im Lauf der Zeit wurden unterschiedliche agile Ansätze in verschiedenen Branchen zur Lieferung erfolgreicher Systeme eingeführt, entwickelt, getestet und freigegeben; zu den bekanntesten gehören Scrum und Kanban.

1.3 Kanban

Kanban ist ein von Toyota entwickeltes, visuelles System, anhand dessen der Projektmanagementworkflow organisiert wird. Im Japanischen bedeutet Kanban „Zeichen“ oder „Tafel“ (Sugimori et al., 1977). In der Vergangenheit wurde in den meisten Branchen nach einem Push-System gearbeitet. Dabei wurden Produkte nach Bedarfsprognosen und nicht nach tatsächlichem Kund:innenbedarf geliefert. Der aktuelle Marktbedarf oder Kund:innenanfragen blieben unberücksichtigt. Dagegen ist Kanban ein **Pull-System** und Teil der agilen Bewegung. Ein Pull-System basiert auf tatsächlichen Kund:innenanfragen und schickt eine Aufgabe erst dann an die Fertigungsline, wenn ein Bedarf für ein bestimmtes Produkt besteht. Kanban wurde für den Einsatz in verschiedenen Industrien modifiziert, was zu weniger Aktivität bei gleichbleibender Produktivität geführt hat. Gleichzeitig stieg bei gleichen Kosten für den Betrieb der Mehrwert für den:die Kund:in. Im

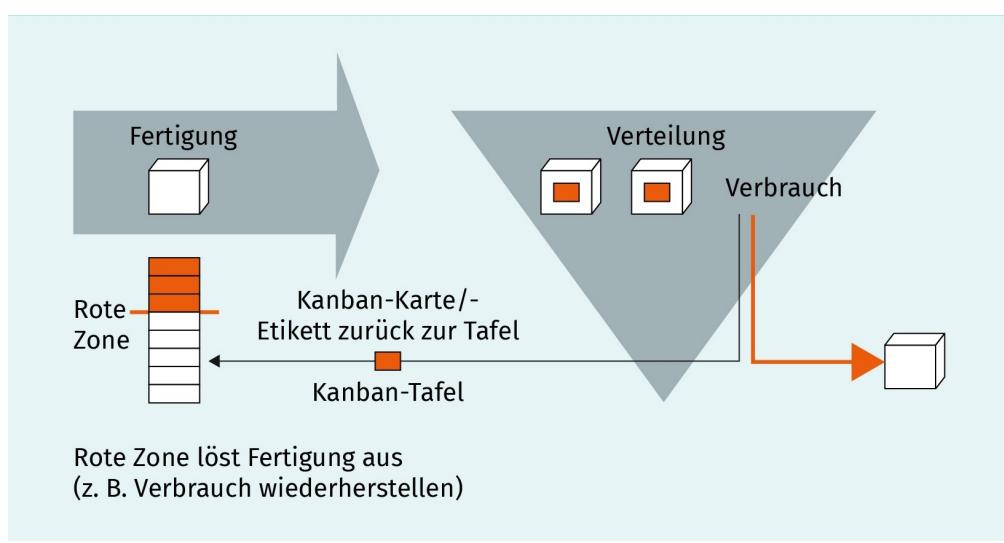
Pull-System

Eine neue Aufgabe wird erst durch eine Kundenanfrage ausgelöst, wodurch in der Produktion Kosten und Material-einsatz optimiert werden.

Mittelpunkt dieser Methode stehen Aufgaben, ihre Ausführung, der Zeitpunkt der Ausführung und der Umfang, diese werden visuell dargestellt. Drei Aspekte sind dabei wichtig: das Kanban-Board, die Kanban-Karten und Swimplanes.

Die folgende Abbildung zeigt das ursprüngliche Kanban-System:

Abbildung 5: Kanban-Struktur



Quelle: Jbarta (2013) in Anlehnung an Waldner (1992).

Eine Kanban-Karte wird im Kanban-Workflow auf ein Verbrauchssignal aus der roten Zone entnommen. Gleichzeitig wird in der Fertigungslinie eine Aufgabe gestartet. Nach der Fertigung wird das Produkt ausgeliefert und die Karte wieder auf der Kanban-Tafel angebracht. Das Board, auf dem alle Aufgaben und ihr jeweiliger Status angezeigt wird, wird als Kanban-Board bezeichnet. Auf dem Kanban-Board befinden sich verschiedene Spalten, die den Aufgabenstatus anzeigen, und deren Anzahl je nach Team- und Projektaufbau angepasst wird. Drei Spalten sind jedoch immer auf dem Board: eine für Projektideen, eine für laufende Arbeiten, und eine für abgeschlossene Arbeiten. Sie könnten zum Beispiel mit „Offen“, „Laufend“ und „Erlledigt“ betitelt werden. Aufgabendetails, wie die Aufgabenentwicklung bzw. der Status, kann das Team den sogenannten Kanban-Karten entnehmen. Fallen Aufgaben in verschiedene Kategorien, können Kanban-Swimlanes zur horizontalen Kategorisierung auf dem Board verwendet werden. Die nachfolgende Abbildung zeigt ein Beispiel für ein Kanban-Board.

Abbildung 6: Online Kanban-Board

Online Kanban-Board			
	Offen +	In Arbeit +	Erledigt +
Team A	Design T1	Datensammlung T1	Validierung T2
	Präsentation T2	Design T3	Datensammlung T3
	Validierung T3		Konstruktion T3
Team B	Validierung T4	Modell 1 Test T4	Datensammlung T4
	A/B-Test T4	Modell 2 Test T4	Analyse T4
		Modell 3 Test T4	

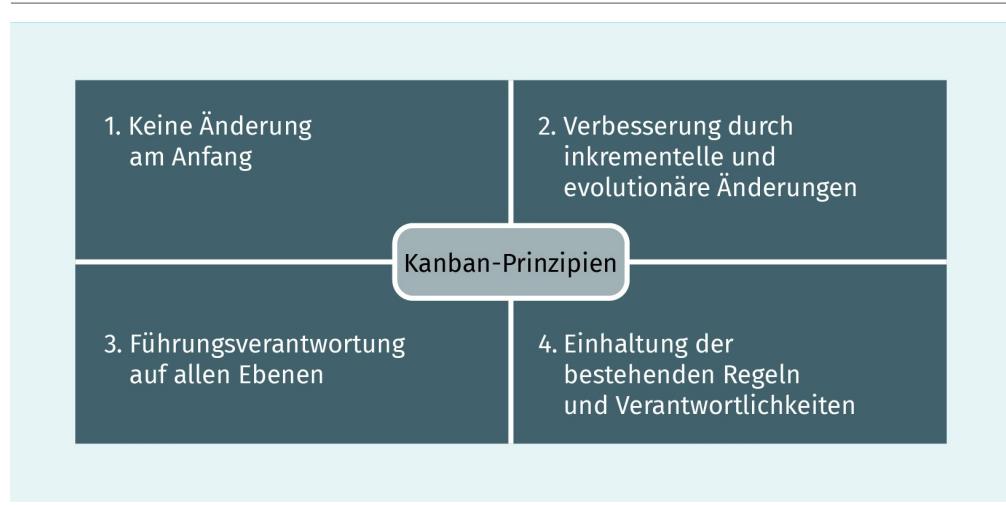
Quelle: Asadi (2020).

Aufgrund des technischen Fortschritts und des wachsenden Einsatzes des Internets bieten viele Online-Plattformen heute Kanban an. Dennoch folgen viele Firmen der traditionellen Kanban-Philosophie, bei der tatsächliche Tafeln bevorzugt werden, da sie den Austausch fördern. Durch das Kanban-Board ist der Arbeitsstatus für das Team transparent, mögliche Engpässe können schnell bestimmt und Hindernisse oder Risiken beseitigt werden.

Grundprinzipien und gängige Praktiken

Toyota hat bei der Entwicklung von Kanban vier Grundprinzipien und sechs gängige Praktiken, die ein Team beim Einsatz von Kanban berücksichtigen sollte, eingeführt (Sugimori et al., 1977). Die vier Prinzipien werden in der folgenden Abbildung dargestellt:

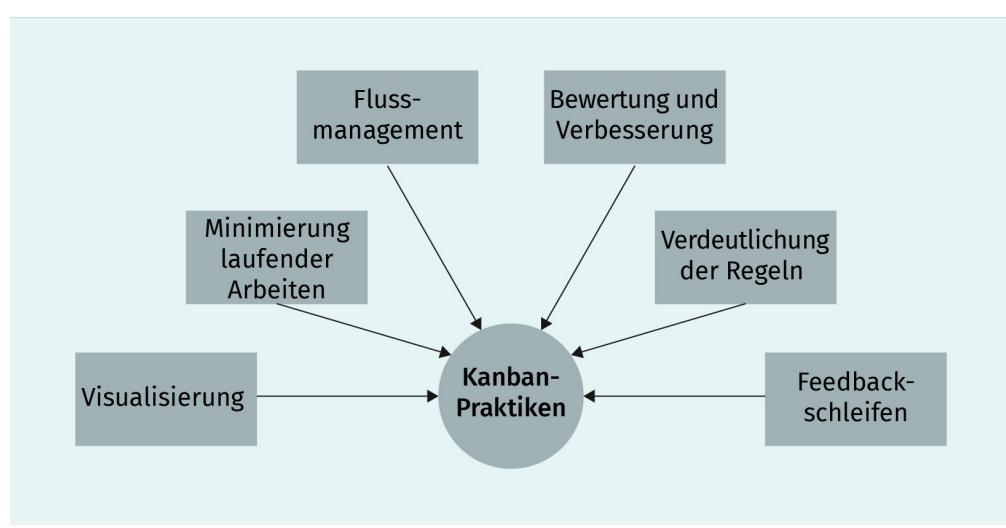
Abbildung 7: Kanban-Prinzipien



Quelle: Asadi (2020).

Diese Prinzipien tragen erheblich zur Verbesserung respektvoller Zusammenarbeit in einem Kanban-Team bei. Wie bereits erwähnt, gehören zur Kanban-Methode auch sechs Praktiken, die zu Beginn eines neuen Kanban-Zyklus berücksichtigt werden sollten und die in der nachstehenden Abbildung veranschaulicht werden:

Abbildung 8: Kanban-Praktiken



Quelle: Asadi (2020).

Wie bereits erklärt, liegt der Schwerpunkt bei Kanban auf der Veranschaulichung von Aufgaben auf dem Kanban-Board. Jede Aufgabe sollte benannt werden und kann zusätzliche Informationen, wie Kategorie, Erstellungsdatum, Fertigstellungstermin, Eigentümer:in oder Anforderungen enthalten. Wenn dem Kanban-Board eine Aufgabe hinzugefügt wird, sollten die Karten, die mit „In Arbeit“ gekennzeichnet sind, eine bestimmte Anzahl nicht überschreiten, damit die Produktivität und die Konzentration gewährleistet werden kann.

Jede Aufgabe wird innerhalb eines Sprints ausgeführt. Falls dies im entsprechenden Inkrement nicht möglich ist, kann sie ins nächste Inkrement übernommen werden. Bei anfänglicher Anwendung der Kanban-Methode ist es nicht leicht, eine geeignete Menge an Aufgaben festzulegen, die sich „In Arbeit“ befinden.

Daher empfiehlt es sich, zunächst mit einer niedrigen bzw. angemessenen Menge zu beginnen, um schneller zu Ergebnissen zu gelangen. Mit der Zeit kann ein Team diese Limits erhöhen, wenn es sich abschätzen lässt, dass alle in der Spalte „In Arbeit“ befindlichen Aufgaben ohne Probleme abgearbeitet werden können. Wenn ein Team innerhalb eines Sprints Zeit hat, andere Tickets zu bearbeiten, kann es eine Aufgabe durch das Pull-System entnehmen und, ohne auf den nächsten Sprint zu warten, mit der Abarbeitung beginnen. Wenn Aufgaben auf dem Kanban-Board verschoben werden, muss auf den Arbeitsfluss geachtet werden, sodass klar bleibt, wer zu welchem Zeitpunkt welche Aufgaben übernimmt. Dies ist besonders im Hinblick auf die Arbeitsaufteilung im Team wichtig. Kanban darf also nicht als Mikromanagement verstanden werden oder zur Überlastung des Teams führen. Es sollte dagegen zu einem transparenten, gesunden und produktiven System führen, in dem ein Team Aufgaben schneller ausführen kann. Daher bietet es sich an, Richtlinien klar darzustellen und Personen bzw. Arbeit zu verbinden. Einen interessanten Aspekt von Kanban stellt das Feedback dar, das Teammitglieder geben dürfen und erhalten, denn dadurch wird das Team agiler. Teams können täglich sogenannte Stand-Up-Meetings von bis zu 15 Minuten vor dem Kanban-Board abhalten, um den Stand abzuklären und einander Feedback zu geben. Zur Verbesserung der Qualität einer Aufgabe oder eines Ergebnisses kann ein Teammitglied einen Review mit dem Ziel durchführen, das Ergebnis am Teamstandard zu messen und den Verantwortlichen der Aufgabe Feedback zu vermitteln.

Vorteile von Kanban

Zur Einführung von Kanban sind keine großen Veränderungen notwendig, da Begriffe, Implementierung und Verfahren auch für Mitarbeitende ohne Vorwissen im Projektmanagement leicht verständlich sind. Durch die Verwendung von agilen Methoden wie Kanban kann sich die Produktivität eines Teams erhöhen. Gleichzeitig behält es den Überblick über die bereits erledigten und die noch offenen Aufgaben. Da alle Aufgaben auf dem Board sichtbar sind, wird das Board gewissermaßen zum Informationszentrum des Teams, auf dem sich alle für die Ausführung einer Aufgabe notwendigen Informationen ablesen lassen. Wenn eine Aufgabe in eine Spalte geschoben wird, sieht man sofort, ob die Spalte bereits überfüllt ist und sich ein Engpass im Workflow bildet. Aufgrund der Richtlinie, wenige Aufgaben mit dem Status „In Arbeit“ zuzulassen, sind Kanban-Teams reaktionsfreudiger und produktiver und daher zufriedener mit ihrer Arbeitsauslastung. Ein Team gewinnt durch Kanban auch an Flexibilität, da Änderungen, z.B. eine neue Aufgabe, direkt auf dem Board vorgenommen werden können und vom nächsten freien Teammitglied priorisiert und gelöst werden können. Durch das schnelle Reagieren auf Änderungen wird das Team produktiver und arbeitet besser zusammen, da alle Teammitglieder einander unterstützen können und in der Lage sind, auch Aufgaben anderer zu übernehmen. Die Hauptvorteile von Kanban lassen sich wie folgt zusammenfassen:

- Kanban ist leicht zu verstehen und umzusetzen.

- Das Team behält anhand des Kanban-Boards den Überblick und kann dort Informationen zu Aufgaben und dem jeweiligen Stand schnell einsehen.
- Alle Änderungen können als neue Aufgaben auf die Liste der offenen Aufgaben des Kanban-Boards gesetzt werden. Dadurch kann sie das Team zügig integrieren.

Agiles Kanban

Agiles Kanban ist eine Kombination aus den Prinzipien der agilen Projektmanagementmethode und Kanban. Bei dieser Methode wird Kanban in Agile integriert, um Prozessworkflows zu managen und Benutzergeschichten umzusetzen. Jede Benutzergeschichte bzw. jede Aufgabe im agilen Prozess entspricht dabei einer Karte auf dem Kanban-Board und wird vom Status „Offen“ über „In Arbeit“ schließlich zu „Erledigt“ geschoben, wenn die Einhaltung aller Richtlinien gewährleistet ist und alle notwendigen Schritte auf dem Board dargestellt wurden. Die teamleitende Person sollte die Anzahl der „In Arbeit“ befindlichen Aufgaben der Fähigkeit des Teams anpassen, sodass während des Entwicklungszyklus produktiv, konzentriert und flexibel gearbeitet werden kann. Die Aufgaben werden nach Kund:innenanforderungen priorisiert, und Änderungen werden angenommen und entsprechend in den Workflow integriert. Der Hauptfokus liegt dank der erhöhten Zusammenarbeit des Teams auf der schnelleren Lieferung des Ergebnisses und dem Mehrwert für die Kund:innen. Kanban kann in allen Industriezweigen oder Projekten, bei denen notwendige Aufgaben vorhersehbar sind und ein Pull-System zur Verfügung steht, eingesetzt werden. Unter diesen Voraussetzungen lässt sich Kanban in einer Organisation problemlos umsetzen. Ein Beispiel wäre ein Data Science Team eines Autoherstellers, das alle Daten bereits zur Verfügung hat und bei dem die technische Abteilung in regelmäßigen Abständen Fragen zur Verbesserung des Produkts stellt, die anhand der Datenquellen beantwortet werden müssen. Jede Frage ist eine Aufgabe auf dem Kanban-Board. Nachdem alle Anforderungen erfüllt wurden und eine Aufgabe abgearbeitet wurde, kann eine neue aus der Spalte der offenen Aufgaben des Kanban-Boards entnommen werden. So können die Fragen der Abteilung nacheinander abgearbeitet werden. Obwohl Kanban eine gute Lösung für Organisation, die mit einem Pull-System arbeiten, ist, eignet sich ein anderer Ansatz womöglich besser für Organisationen, die mit einem Push-System und komplexeren Aufgaben und mit vielen Stakeholder:innen arbeiten. Händler:innen arbeiten beispielsweise mit Push-Systemen. Ein neues Produkt wird anhand des Kund:innenbedarfs für andere Produkte auf dem Markt eingeführt und daraufhin durch entsprechendes Marketing gefördert.

1.4 Scrum

Scrum ist eine Projektmanagementmethode, die von Hirotaka Takeuchi und Ikujiro Nonaka im Harvard Business Review vorgestellt wurde. In ihrem Artikel verwenden die Autoren den Vergleich zum Rugby, um die Vorteile eines strukturierten und selbstorganisierten Teams und die Art und Weise zu beschreiben, wie man diese Teams innovativer und produktiver macht (Takeuchi & Nonaka, 1986). Scrum ist ein Begriff aus dem Rugby und beschreibt, wie ein Team einen Kreis bildet, um den Ball zu erlangen. Die deutsche Bezeichnung dieser Formation ist Gedränge. Folgende Abbildung zeigt ein Beispiel dafür:

Abbildung 9: Gedränge (engl. Scrum) beim Rugby



Quelle: Baucherel (2019).

Die Methode des Agile Scrum wurde von Schwaber und Babatunde am DuPont Forschungszentrum und der Universität von Delaware in der Softwareentwicklung eingeführt. Sie wurde zunächst für die Teamführung verwendet und später in der Easel Software Development Corporation angewandt (Sutherland, 2004). Entwicklungsteams durchlaufen dabei Iterationsschleifen, um inkrementell Produktergebnisse zu erzielen. Agile Scrum liefert dazu eine Strategie, die ein Team befolgen kann, um die Produktfunktion zu erlernen, Erfahrungen einzusetzen, Wissen zu erlangen und aufgrund von Änderungen Anpassungen vorzunehmen, um hochwertige Ergebnisse zu erzielen. Dabei können sich Teammitglieder selbst organisieren, die Qualität ihrer Arbeit durch regelmäßigen Austausch miteinander und mit anderen Stakeholder:innen verbessern sowie aufkommende Probleme gemeinsam lösen. Dieses Modell eignet sich am besten für Firmen oder Teams, deren Produktentwicklungsprozess in zwei- bis vierwöchige Iterationen unterteilt werden kann.

Das Agile Scrum Framework

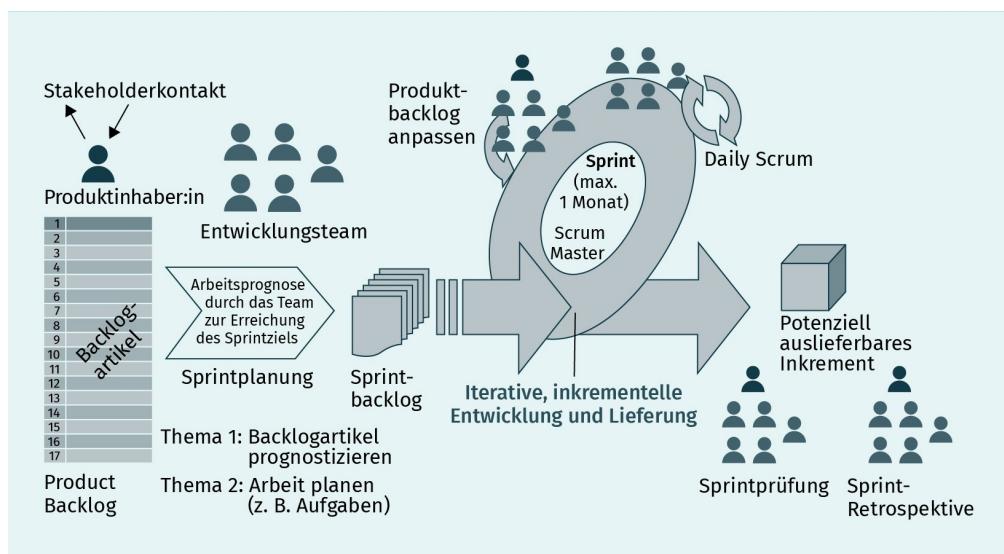
Um ein Agile Scrum Framework zu erstellen, sollten den Teammitgliedern die ursprünglichen Prinzipien bekannt sein (Takeuchi & Nonaka, 1986).

- **Transparenz:** Die Arbeitsumgebung und -kultur sollte den Austausch von Wissen über den Prozess und die inhärenten Hindernisse ermöglichen, sodass Probleme sofort gelöst werden können.

- **Überprüfung:** Um die Qualität des Endprodukts zu verbessern und den Prozessablauf zu verstehen, müssen regelmäßig Aufgaben zur Bewertung bestehen und das Team muss Feedback gegenüber aufgeschlossen sein.
- **Anpassung:** Es muss eine ständige Prüfung stattfinden, anhand derer Dinge, die nicht zum Projekt Mehrwert beitragen, eliminiert werden können.

In der nachstehenden Abbildung werden verschiedene Elemente und Rollen aufgeführt, die zu einem erfolgreichen Scrum Framework gehören:

Abbildung 10: Agile Scrum Framework



Quelle: Mitchell (2015).

Sprints und Agile Scrum

Ein Projekt und dessen Umfang können sich bei Agile Scrum ändern, die Kosten und der Zeitrahmen bleiben jedoch gleich. Die Projektlänge ist je nach notwendigem Entwicklungsaufwand und nötiger Entwicklungszeit in gleiche Abschnitte, sogenannte Sprints, aufgeteilt. Wie beim agilen Sprint kann ein Sprint bei Agile Scrum selbst zum Projekt werden, an dessen Ende ein Produkt zur Kund:innenprüfung steht. Die zum Abschluss eines Sprints notwendige Zeit sollte gleichbleiben, sodass Kosten genau prognostiziert werden können. Im Idealfall dauert ein Sprint ein bis zwei Wochen. Bei einem zweiwöchigen Sprint betragen die Kosten für die Mitarbeitenden beispielsweise die Hälfte der Monatsgehalter.

Rollen und Agile Scrum

Bei Agile Scrum gibt es verschiedene Rollen, die Teammitglieder aufgrund ihrer Erfahrung annehmen können und die für den Erfolg eines Teams wichtig sind (Sims & Johnson, 2012). Dazu gehört die Rolle des:der Produkt:inhaberin, die praktisch dieselbe ist, wie die des Product Owners bei Agile.

- Produktinhaber:innen sind für die Steuerung des Produktbacklogs verantwortlich, sodass das erwünschte Ergebnis und die Kund:innenzufriedenheit gewährleistet werden können.
- Der Scrum Master stellt sicher, dass ein Team die Werte und Prinzipien von Agile befolgt und dass am Ende des Sprints dessen Ziel erreicht wurde. Scrum Master leiten Teams nicht direkt, sondern unterstützen und fördern es, sodass alle Anforderungen zur erfolgreichen Ausführung der Aufgaben erfüllt sind. Während des Sprints vertreten Scrum Master das Team und arbeiten eng mit den Produktinhabern zusammen.
- Die Rolle des:der Entwickler:in eignet sich hervorragend für passionierte Produktentwickler:innen im Team, die die notwendigen Fähigkeiten zur Lieferung eines qualitativ hochwertigen Produkts und zur Kund:innenzufriedenheit mitbringen.

Artefakte als essenzielle Elemente des Agile Scrum

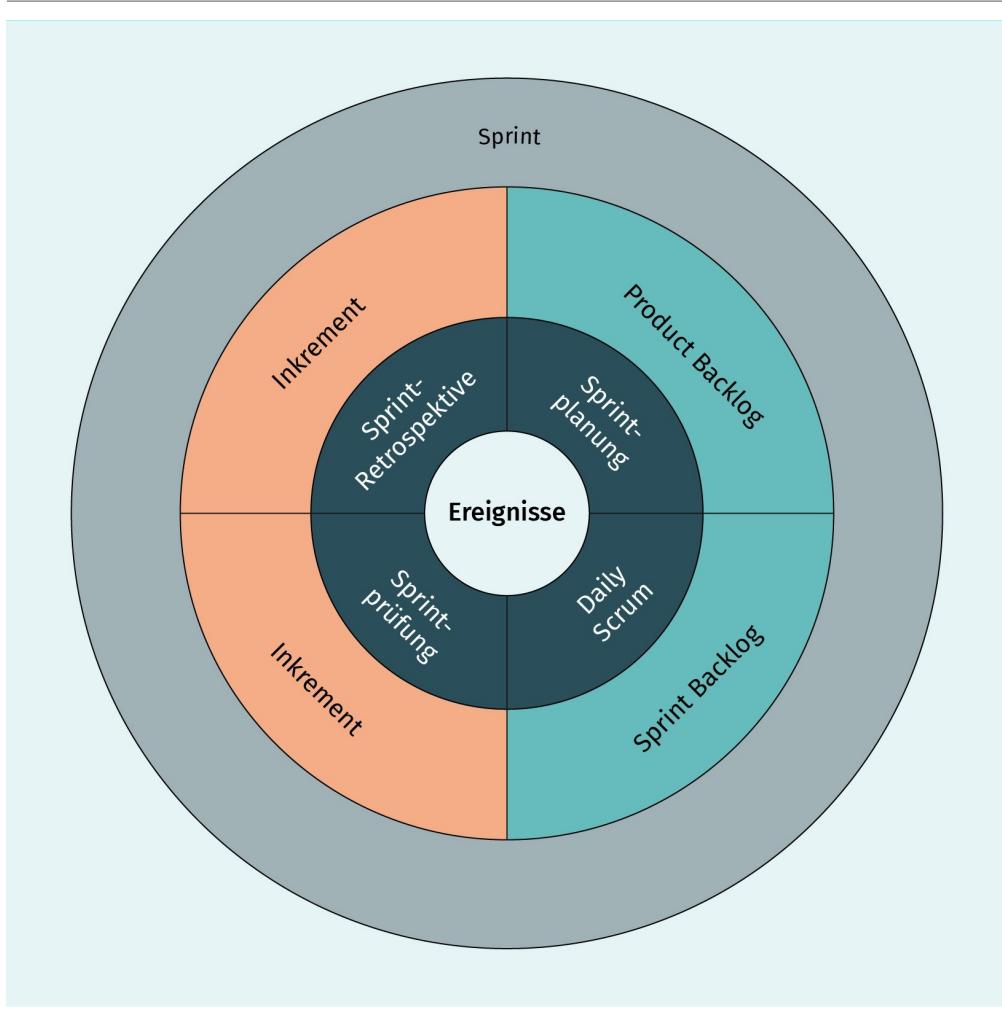
Neben der Festlegung der Rollen sind für das Agile Scrum Framework auch Werkzeuge, die sogenannten Artefakte, wichtig (Jongerius et al., 2013):

- Das Product Backlog ist eine geordnete Auflistung von Anforderungen, die zur Umsetzung von Projektänderungen gebraucht werden. Die einzelnen Artikel in einem Product Backlog werden als Benutzergeschichten bezeichnet. Die Pflege des Product Backlogs unterliegt dem Product Owner, der auch den Geschäftswert der einzelnen Artikel bestimmt. Artikel in einem Product Backlog sind Optionen, die ein Entwicklungsteam wählen und im nächsten Sprint abarbeiten kann. Das heißt allerdings nicht, dass alle Artikel in einem Product Backlog geliefert werden.
- Das Sprint Backlog enthält eine Artikelliste, die vom Entwicklungsteam zur Erfüllung des Sprintziels bearbeitet werden sollte. Ein Element oder eine Benutzergeschichte kann auch in kleinere Aufgaben geteilt werden. Jede Benutzergeschichte bekommt dabei eine:n Inhaber:in. Diese:r Inhaber:in kann durch andere im Team bei der Ausführung der Aufgaben in der Benutzergeschichte unterstützt werden. Ein Entwicklungsteam sollte abschätzen können, welche Artikel es in einem Sprint liefern kann, da der Product Owner dem Sprint Backlog nach Beginn des Sprints keine Artikel mehr hinzufügen kann.
- Ein Inkrement ist eine Sammlung an Produktelementen, die verlässliches und funktionsfähiges Output des Sprints darstellen könnten. Wenn ein Inkrement abgeschlossen wird, gilt es als „Erledigt“ und kann dem:der Kund:in geliefert werden.
- Die „Definition von Erledigt“ wird vorher vom Team anhand von Standards und Regeln festgelegt. Diese werden verwendet, wenn Artikel zum Sprint Backlog hinzugefügt und während des Sprints abgearbeitet werden.

Agile Scrum-Ereignisse

Während eines großen Sprintzyklus müssen bestimmte Ereignisse stattfinden. Diese Ereignisse und die Beziehungen zu den agilen Artefakten werden in folgender Abbildung dargestellt:

Abbildung 11: Agile Scrum-Ereignisse



Quelle: Asadi (2020).

- **Sprintplanung:** Dabei handelt es sich um eine Besprechung am Anfang jedes Sprints, um einen Plan aufzustellen und die Aufgaben, die im Sprint ausgeführt werden, festzulegen. Dies dauert normalerweise über eine Stunde. Dabei können Teams Artikel aus dem Product Backlog auswählen und in das Sprint Backlog übernehmen. Nach diesem Schritt entscheidet der Product Owner gemeinsam mit dem Team, welche Artikel ausgewählt werden. Das Entwicklungsteam bestimmt daraufhin die Akzeptanzkriterien für das Ergebnis und die Lieferung des Produkts.
- **Sprint Review:** Am Ende des Sprints geht das gesamte Team die Aufgaben noch einmal durch, die im letzten Sprint Backlog ausgeführt wurden. Stakeholder:innen und Kund:innen, die am Projekt teilnehmen, können diesen Besprechungen beiwohnen, um auf den neuesten Stand zu kommen, Feedback ans Entwicklungsteam weiterzuleiten oder um Änderungen am Produkt vorzubringen. Diese Änderungen können daraufhin dem Backlog zukünftiger Sprints als neue Artikel hinzugefügt werden.

- **Sprint-Retrospektive:** Am Ende eines Sprint Reviews können das Entwicklungsteam und der Product Owner den Sprint, die positiven und die negativen Elemente oder den Änderungsbedarf an Standards oder Inkrementen besprechen. Diese Besprechung trägt zum Verständnis der Anliegen des Entwicklungsteams bei und führt zur Umsetzung des Feedbacks in den nächsten Sprints, was zu einer gesteigerten Produktivität führt.

Vor- und Nachteile von Agile Scrum

Agile Scrum eignet sich besonders für Branchen mit komplexen Projektprozessen, die Ergebnisse bewerten müssen und die um Kund:innenzufriedenheit bemüht sind (Jongerius et al., 2013). Am meisten profitierte die Softwarebranche von dieser Methode und vor allem kleinere Projektteams von maximal zehn Personen. Agile Scrum wird jedoch auch in vielen anderen Branchen eingesetzt (z.B. Finanzdienstleistung, Ingenieurwesen, Bau), da diese Methode für anpassungsfähige und flexible Projektprozesse gedacht ist, die während der Entwicklung eventuell geändert werden müssen (Jongerius et al., 2013). Obwohl Organisationen mit Agile Scrum mehr Einsatzbereitschaft, Mut, Fokus und Respekt verzeichnen können, gibt es auch Nachteile. Agile Scrum bietet sich nicht an, wenn Teammitglieder Tele- oder Teilzeitarbeit leisten, wenn besondere Fertigkeiten verlangt werden, oder wenn viele Stakeholder:innen oder externe Abhängigkeiten vorhanden sind. Unter obigen Voraussetzungen sollte ein Team dennoch überlegen, ob es Agile Scrum eingesetzt möchte, denn diese Methode kann stets an die individuellen Anforderungen eines Teams angepasst werden, womit sich diese Nachteile möglicherweise kompensieren lassen.

1.5 Andere moderne Methoden

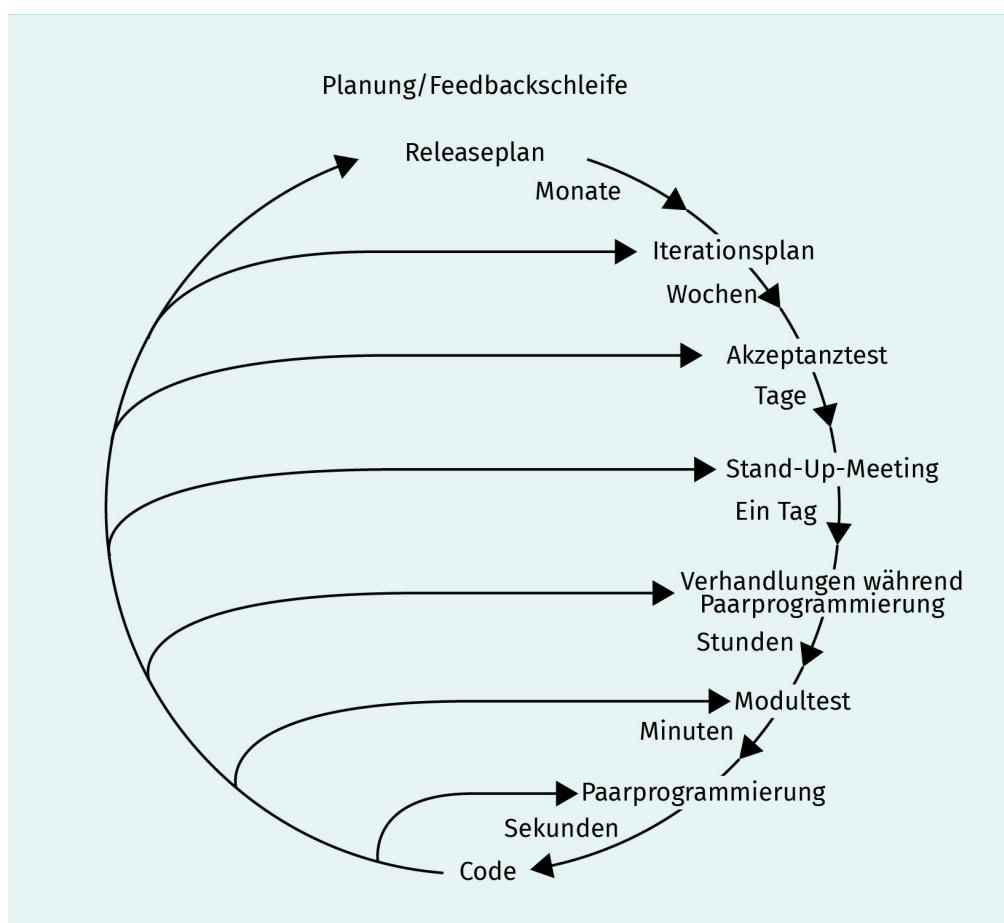
Seit Jahrzehnten hat sich durch klassische und agile Managementmodelle eine Grundlage für die Projektmanagementbedürfnisse von Organisationen herausgebildet. Um diese Methoden über den Rahmen eines einzelnen Teams hinaus einzusetzen, bedarf es einer umfassenden Kenntnis der Ressourcen, Anforderungen und Fähigkeiten einer Firma, sodass Projekte abgeschlossen und damit gute Resultate erzielt werden können. Da das Projektmanagement auf den Abschluss hochwertiger Projekte innerhalb eines bestimmten Zeitrahmens und Budgets abzielt, eignet sich ein Ansatz meistens besser als ein anderer. Um die richtige Entscheidung für einen Projektmanagementansatz zu treffen, benötigen Teams einen guten Überblick über die verschiedenen Methoden und ihre Vorteile. Im folgenden Abschnitt werden einige professionelle Projektmanagementlösungen näher ausgeführt.

Extreme Programming (XP)

Dieses auf agilen Methoden basierende Modell wurde von einem Softwareentwickler von Chrysler im Zuge eines Lohnbuchhaltungsprojekts entwickelt. Es zielt darauf ab, die Qualität eines Projekts durch die schnelle Anpassung an Änderungen zu verbessern (Rosenberg & Stephens, 2008). Durch dieses Modell werden vor allem dann gute Resultate erzielt und eine kollaborative Umgebung geschaffen, wenn sich Anforderungen ändern und ständiges Feedback notwendig ist. Das XP-Modell beruht auf vier Grundaktivitäten: umfangreiches

Codieren, Modultesten, Kund:innen zuhören und Freigabe. Die Anwendung dieser Prinzipien während der Iterationsschleife eines Projektentwicklungsprozesses ermöglicht es dem Entwicklungsteam, den Umfang und das Projekt schnell anzupassen. Folgende Abbildung zeigt die Iterationsstruktur dieser Prinzipien:

Abbildung 12: Extreme Programming



Quelle: Wells (2013).

Daraus geht hervor, dass Kund:innen während des kurzen Entwicklungszyklus Input geben, wodurch das Team den Projektumfang entsprechend anpassen kann. Dieses Modell ist in Softwareunternehmen und der Softwarebranche erfolgreich und wird dort häufig eingesetzt (Newkirk & Martin, 2000).

Adaptive Project Framework (APF-Methode)

Klassische Projektmanagementmethoden lassen sich nicht leicht an Veränderungen und Kund:innenfeedback anpassen. Dazu muss eine Organisation die Projektmanagementmethodik anhand eines soliden Projektplans prozess- und phasenübergreifend anpassen. Das ist bei klassischen Modellen nicht möglich. Große Änderungen sind heute allerdings unvermeidlich und Betriebe müssen ihre Methodik an die Projektziele anpassen, um

Erfolg zu erzielen. Die APF-Methode eignet sich hervorragend dazu, zunächst unbekannte Faktoren, die im Lauf einer Projektphase auftauchen können, einzubeziehen. Dabei können Projektfunktionen, -unterfunktionen, Anforderungen und Eigenschaften dokumentiert werden, bevor die Projektziele festgelegt werden. Das bedeutet, dass ein Entwicklungsteam in Kommunikationszyklen und nicht nach Sprints arbeitet. Kund:innen und Stakeholder:innen können in jedem Zyklus Änderungen am Projektumfang vorbringen, die ein:e Entwickler:in überdenkt und gegebenenfalls in der neuen Phase implementiert. Obwohl Kund:innen und Stakeholder:innen bei der APF-Methode enger mit dem Team zusammenarbeiten und den Projekterfolg mitbestimmen können, kann der Fokus auf die Entwicklungsphase darunter leiden.

Six Sigma

In den vergangenen Jahren haben Firmen versucht, die Kapazität ihrer Geschäftsprozesse mithilfe dieser Methode zu verbessern. Sie wurde von einem Ingenieur bei Motorola entwickelt, der datengetriebene Methodiken im Projektmanagement einführen wollte (Mehjerdi, 2011). Die Hauptphilosophie von Six Sigma beruht auf Messung, Analyse, Verbesserung und Steuerung der Prozesse, die auf das Input wirken, um ein hochwertiges Output zu erhalten. Durch die Inputsteuerung sind Organisationen in der Lage, ein Projekt erfolgreich zu liefern, denn dadurch wird der Prozess vorhersehbar und die Ursachen für Misserfolge während der Projektentwicklung beseitigt. Die Prozessvariationen werden minimiert und Ergebnisse können zuverlässig und zu niedrigeren Kosten erzielt werden. Diese Methode erfordert erfolgreiche Strukturen, Einsatz und Fähigkeiten von der Leitungsebene bis zum:zur Entwickler:in. Ein Mangel an Wissen oder Verantwortung kann große Auswirkungen auf das Output nach sich ziehen. Der einzige Nachteil besteht darin, dass bei Benutzer:innen das Gefühl von Mikromanagement aufkommen kann, da Steuerung ein wesentlicher Bestandteil dieser Methode ist.

Scrumban

Mit der Weiterentwicklung von Kanban entstand Scrumban, um Scrum-Teams dabei zu helfen, die Vorteile von Kanban zu entdecken und zu nutzen (Nikitina et al., 2012). Die Struktur ähnelt der Scrum-Methode, die Arbeitskultur basiert auf Kanban. Bei Scrumban wird ein visuelles Kanban-Board verwendet und es werden kleine Iterationen organisiert, ähnlich wie beim Scrum Sprint. Dadurch können Änderungen schnell ins Projekt integriert werden. Die hervorragende Kombination aus agilem Scrum und dem Pull-System von Kanban führt zu stark verbesserten Prozessen und Ergebnissen. Der beim agilen Scrum verwendete Prozess findet auch bei Scrumban statt, sodass ein Team besser zusammenarbeiten kann und den Stand des Projekts nicht aus den Augen verliert. Statt der Sprintplanung beim agilen Scrum wird bei Scrumban die verfügbare Kapazität dadurch gefüllt, dass dem Kanban-Board weitere Karten hinzugefügt werden. Es wird visualisiert, was sich „in Arbeit“ befindet, um die Produktivität des Teams zu steigern.

Daher eignet sich diese Methode besonders für Organisationen, die sich um laufende Projekte kümmern müssen oder für die Scrum in der ursprünglichen Form nicht flexibel genug ist.

Scrum of Scrum

Dieses maßgerechte Scrum-Modell wird dazu verwendet, um verschiedene Geschäftsbereiche mit mehreren Produktlinien zu koordinieren und um Teams miteinander zu verbinden. Auf diese Weise sollen komplexe Lösungen gefunden werden. Scrum of Scrum findet in verschiedenen Organisationen und Branchen, z. B. in der Software-Branche, Einsatz (Mundra et al., 2013). Diese Methode führt leistungsstarke Scrum-Teams zusammen und ermöglicht es ihnen, gemeinsam an den Projektzielen zu arbeiten. Dies führt zu gegenseitigem Respekt und Vertrauen unter den Teammitgliedern und dazu, dass sie sich an den Projektprozessen beteiligen können. Geeignet sind daher Teams mit maximal sechs Mitgliedern, denn größere Teams können die Kommunikation und Produktbereitstellung erschweren, während die Aufteilung in kleinere Teams gute Strukturen schafft und zu besseren Resultaten führt. Bei der durch Scrum of Scrum geschaffenen Struktur wird jedem Team vom Product Owner eine Kernaufgabe zugewiesen, der sogenannten Organisationslieferung. Daher sind die Produktinhaber:innen für den Informationsfluss zwischen den Teams in Hinblick auf den Bedarf und den Umfang verantwortlich. Der:Die Hauptproduktinhaber:in koordiniert die Zusammenarbeit der Produktinhaber:innen der verschiedenen Teams und leitet sie anhand der Produktvision an (Mundra et al., 2013). Product Owner nehmen täglich an Stand-Up-Meetings teil, in denen die Sprintziele, die Hindernisse und Verbesserungen bezüglich anderer Projekte, für die ihr Team verantwortlich ist, besprochen werden.

Lean Management

Das sogenannte Lean Management gewinnt im Rahmen des Projektmanagements an Bedeutung, denn diese Methode kann auf alle Geschäftssituationen oder Produktkonzepte angewandt werden. In Firmen, die dieses Modell einsetzen, kommt es zu einer höheren Leistung und steigenden Werten (Ballard & Howell, 2003). Das Modell beruht auf drei Hauptregeln:

- Mehrwert für Kund:innen schaffen.
- Verschwendungen oder Prozesse ohne Mehrwert für das Endprodukt oder Vorteil für Kund:innen abschaffen.
- Produkte und Prozesse kontinuierlich verbessern.

Bei dieser Methode geht es nicht um den Projektentwicklungsprozess, sondern um den:die Kund:in, die Verbesserung der Arbeitsprozesse und die gesteckten Ziele. Dabei sollte die Verantwortung unter den Teammitgliedern aufgeteilt werden, und Teamleiter:innen sollten Führungsaufgaben teilen. Diese Methode ermöglichte es, Prozesse abzuschaffen oder zu kürzen und Kosten zu senken. Gleichzeitig wurden Projektziele schneller oder besser erreicht (Ballard & Howell, 2003). In der digitalen Welt wurden Softwareentwicklungs- und Startup-Methoden nach dem Prinzip des Lean Management entwickelt. Aus der Perspektive des Betriebs kann durch Lean Management der Projektentwicklungszyklus verkürzt und der Geschäftswert gemessen werden. Lean Management beruht auf den fünf, in der folgenden Abbildung aufgeführten Prinzipien.

Abbildung 13: Prinzipien des Lean Management



Quelle: Asadi (2020).

An erster Stelle steht die Wertfindung durch die Analyse des Kund:innenbedarfs. Die Werte der Kund:innen werden daraufhin auf den Wertfluss des Unternehmens projiziert und es wird der Workflow konzipiert. Ein guter Workflow zeichnet sich dadurch aus, dass anhand eines Pull-Systems Arbeit nur dann angefordert wird, wenn ein entsprechender Bedarf besteht. Dadurch können Ressourcen optimiert werden. In einer Pizzeria wird beispielsweise nur dann eine Pizza zubereitet, wenn eine Bestellung dafür vorliegt. Um gute Ergebnisse anhand eines stabilen Workflows zu erzielen, muss ein Team diesen ständig anhand des Kund:innenbedarfs verbessern. Das Lean Management liefert nützliche Richtlinien zum Aufbau eines standfesten Teams und hilft dabei, Kund:innenänderungen zu definieren und Hindernisse schnell abzubauen.

1.6 Einführung von Agile

Das schnelle Wachstum des Technologiesektors und des Markts kann zu Veränderungen der Infrastruktur einer Firma führen. Auch die Art und Weise, wie auf Marktanfragen reagiert wird, muss möglicherweise angepasst werden. Dabei ist es für manche Firmen nicht einfach, alte Strukturen und Produktmanagementstrategien zu ändern. Wenn beispielsweise ein Hardware- oder Automobilhersteller Software im Haus herstellen will, muss man bei der Umstellung vom klassischen Projektmanagement auf Agile mit Herausforderungen rechnen. Die Vorteile des klassischen Projektmanagements sind Stabilität, eine feste Prognose und Kostenschätzungen. Für die agile Methode ist dagegen mehr Flexibilität notwen-

dig und man muss mit Risiken aufgrund der hohen Änderungsrate rechnen. Wenn eine Firma jedoch auf dem Markt bestehen will, muss sie unter Umständen auf Agile umstellen und ein passendes Modell finden.

In der folgenden Tabelle werden das agile und das klassische Projektmanagement einander gegenübergestellt:

Tabelle 1: Vergleich der Methoden des agilen und des klassischen Projektmanagements

	Klassisches Projektmanagement	Agiles Projektmanagement
Anforderungen	klare Anforderungen und wenige Änderungen	klare Anforderungen und viele Änderungen
Kund:in	nicht am Prozess beteiligt	enge Zusammenarbeit
Dokumentation	formale Dokumentation	implizites Wissen
Größenordnung des Projekts	groß	klein bzw. mittelgroß
Organisationsstruktur	linear	iterativ
Modellpräferenzen	Anpassung an Änderungen	Änderungen im Voraus zu bedenken

Quelle: Asadi (2020).

Wenn Projekte nicht termingerecht, mit entsprechendem Standard und unter Erfüllung der Akzeptanzkriterien des Teams oder der Kund:innen geliefert werden können, sollte untersucht werden, ob der Projektmanagementansatz womöglich für diese Mängel verantwortlich ist. Klassisches Projektmanagement kann unnötig komplex und uneffektiv sein. Die Einführung eines neuen, verbesserten Systems, wie Agile, kann einem Team wieder zu Erfolg verhelfen. Für Teams im Bereich der Datenanalyse oder Data Science könnte eine Mischung aus verschiedenen agilen Ansätzen, wie z. B. Scrumban, das beste Modell sein. Kommt diese Methodik zum Einsatz, kann das Team Änderungen, direkt nachdem von Kund:innen Feedback eingegangen ist, umsetzen. Die Durchführung von Änderungen kostet wahrscheinlich dennoch Zeit, da möglicherweise mehr Daten notwendig sind oder die Kund:in den richtigen Augenblick abwarten muss, um entsprechendes Wissen einzubringen. In der Softwareentwicklung können Änderungen dagegen relativ leicht umgesetzt werden, da selbst ohne Kund:inneninput gesamte Anwendungen geändert oder neue Methoden bzw. Module eingeführt werden können. Unabhängig vom Firmenbedarf sollte klar sein, dass nur ein sowohl für das Projektteam als auch für die Kund:innen passendes Modell notwendig ist. Eine plötzliche Umstellung vom klassischen auf agiles Projektmanagement kann ein Team besonders fordern, besonders, wenn die Teammitglieder nicht mit der Methodik vertraut sind und daher einen völlig neuen Ansatz erlernen müssen. Ein Unternehmen muss Zeit und Geld investieren, um das Team bei der Umstellung zu unterstützen. Zudem müssen Regeln, Ereignisse und Verantwortungen für die Einführung einer neuen Sprach- und Arbeitskultur angepasst werden. Insgesamt bedarf es für einen reibungslosen Übergang von einem alten zu einem neuen Projektmanagementsystem wie Agile, vieler neuer Fähigkeiten. Zudem ist es notwendig, geduldig vorzugehen. Trotz des Aufwands kann sich der Versuch auszahlen.



ZUSAMMENFASSUNG

Diese Lektion bot einen Überblick über die verschiedenen Projektmanagementansätze zur Entwicklung hochwertiger Produkte. Zu Anfang wurde das klassische Projektmanagement behandelt. Dieses Modell ist leicht verständlich, eignet sich besonders für vor Ort oder in Telearbeit entwickelte Projekte und führt zu einer gesteigerten Kund:innenzufriedenheit durch Erreichung der Projektziele. Diese Methode ist jedoch linear und die Terminplanung erweist sich als schwierig. Kund:innenänderungen können nicht integriert werden und die Kreativität der Teammitglieder wird durch den festen Zeitplan gehemmt. Weiterhin wurde das Framework des agilen Projektmanagements, die Rollen, Prinzipien und Eigenschaften thematisiert. Agile hat kürzere Entwicklungsphasen, benötigt zu Projektbeginn weniger Details, beruht aber auf effektiver Kommunikation zwischen Entwickler:innen und Kund:innen und führt zu einer höheren Produktivität des Entwicklungsteams. Anschließend wurde Kanban erörtert. Bei dieser Methode kommen Kanban-Boards zur Visualisierung der Aufgaben zum Einsatz. Kanban hat einen einfachen Workflow, bei dem Änderungen und Kund:innenwünsche akzeptiert werden, wodurch das Entwicklungsteam viel Zeit spart. Eine weitere Methode des agilen Projektmanagements, die besprochen wurde, ist Scrum. Scrum basiert auf einem iterativen Entwicklungsprozess, dem Sprint, bei dem alle Aufgaben zwischen drei Hauptrollen verteilt werden: Produktinhaber:in, Scrum Master und Entwickler:in. Ein Team kann durch Scrum produktiver arbeiten und Kund:innenänderungen am Ende eines Sprints integrieren. Auch die Philosophie, die Vor- und Nachteile von Methoden wie Extreme Programming, Adaptive Project Framework, Six Sigma, Scrumban, Scrum of Scrum, und Lean Management wurden besprochen. Zum Schluss wurden die Vorteile einer Umstellung auf agile Methoden bearbeitet. Dazu gehören die verbesserte Kooperation, Vorteile für kleine und mittlere Projekte und erhöhte Kund:innenzufriedenheit aufgrund der möglichen Integration von Änderungswünschen der Kund:innen während der iterativen Entwicklungsphase.

LEKTION 2

DEVOPS

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Begriffe DevOps und Lifecycle Management zu verstehen.
- den Unterschied zwischen DevOps und klassischem Lifecycle Management zu verstehen.
- die Ziele von DevOps zu verstehen.
- die Auswirkungen von DevOps auf Teams und die Entwicklungsstruktur zu verstehen.
- zu verstehen, wie eine DevOps-Infrastruktur entsteht.
- zu verstehen, wie skalierbare Umgebungen entstehen.

2. DEVOPS

Einführung

Digitale Geschäftsmodelle und -prozesse sowie Dienstleistungen der künstlichen Intelligenz, sogenannte Smart Services, beruhen weitestgehend auf Informationstechnologie (IT) und Software. Aufgrund der Dynamik der heutigen Welt ändern sich Anforderungen und Bedingungen äußerst häufig und zudem sehr kurzfristig. Daher wird die enge Zusammenarbeit und Integration zwischen Softwareentwicklung und IT-Betrieb mit dem Geschäftsbetrieb zu einem immer wichtigeren Wettbewerbsfaktor.

Der Begriff DevOps setzt sich zusammen aus den englischen Abkürzungen für Entwicklung (*Dev* von *Development*), hier speziell die Softwareentwicklung, und Betrieb (*Ops* von *Operations*), wobei hier der IT-Betrieb gemeint ist. Der Zusammenschluss sollte zu Prozessverbesserungen in den Bereichen der Softwareentwicklung und der Systemverwaltung führen. Bei dieser Firmenkultur werden Werte wie Zusammenarbeit, Experimentierfreudigkeit und Lernbereitschaft geschätzt. Das Ziel dabei ist die effektivere und effizientere Zusammenarbeit dreier Abteilungen: Entwicklung, IT-Betrieb und Qualitätssicherung. Diese Abteilungen verfolgen das gemeinsame Ziel der schnellen Implementierung von stabiler und hochwertiger Software für alle Schritte von der Konzeption bis hin zur Auslieferung an die Kundschaft oder die Benutzer:innen (Amazon, o. D.-a). DevOps ist weder Werkzeug noch Software, Technologie, Methodik oder Prozess. Es ist vielmehr als eine Unternehmenskultur mit Prinzipien zu verstehen, die ein Unternehmen langfristig umsetzen möchte.

Bevor wir tiefer in die Thematik von DevOps einsteigen, muss der Begriff des Application Lifecycle Management, der eng mit dem Begriff DevOps zusammenhängt, behandelt werden. Mit Application Lifecycle Management, kurz ALM, wird der Lebenszyklus von einer Idee über ihre Umsetzung bis zur Lösung beschrieben. Durch iterative Zyklen werden Mitarbeitendenzusammenarbeit, Rollen, Prozesse und Information kontinuierlich verbessert. Im Application Lifecycle Management werden mehrere Disziplinen, die im Rahmen zuvor angewandter Entwicklungsprozesse oft separat behandelt wurden, zusammengefasst: Projektmanagement, Anforderungsmanagement, Softwareentwicklung, Testen, Qualitätssicherung, Bereitstellung und Wartung. DevOps wird durch das Application Lifecycle Management unterstützt, indem all diese Disziplinen integriert werden und dadurch die Zusammenarbeit verschiedener Teams innerhalb einer Organisation sehr viel effizienter gestaltet werden kann (Chappell, 2008). Man unterscheidet im Allgemeinen zwischen zwei Ansätzen des Application Lifecycle Management, nämlich dem klassischen ALM und DevOps. In den folgenden Abschnitten werden die Unterschiede zwischen dem klassischen ALM und DevOps erläutert.

2.1 Klassisches Lifecycle Management

Im klassischen Lifecycle Management operieren die Rollen in einer Organisation getrennt voneinander (Shaikh, 2017). Entwickler:innen kommen mit Änderungen gut zurecht, diese stellen einen stetigen Bestandteil ihrer Aufgaben dar. Um auf anspruchsvolle Änderungen eingehen zu können, benötigt eine Firma Entwickler:innen, die Innovation und Änderung anstreben. Im Betrieb sind Änderungen eher hinderlich. Ein Unternehmen benötigt eine Betriebsabteilung zur reibungslosen Abwicklung von Abläufen und Lieferung von gewinnbringenden Leistungen. Betriebsabteilungen lehnen Änderung zu Recht ab, da dadurch die Stabilität und Betriebssicherheit des Unternehmens gefährdet werden.

Entwickler:innen

Entwickler:innen schreiben Code und kümmern sich um Programmfehler. Nach der Bereitstellung einer Anwendung in der Produktionsumgebung sind sie generell interessiert daran, dass die Anwendung funktioniert und dass Feedback von Kund:innen eingeholt wird, sodass Änderungen und Updates zur Verbesserung der Software vorgenommen werden können. Der Zweck der Entwicklungsabteilung ist es, laufend neue Funktionen zu erstellen und bestehende zu verbessern. Die Hauptaufgabe eines:einer Entwickler:in besteht also darin, regelmäßig Änderungen in der Arbeitsumgebung vorzunehmen. Die Rolle traditioneller Entwickler:innen lässt sich anhand der Fähigkeit beschreiben, Änderungen herbeizuführen. Ihr Wert für die Organisation wird in aller Regel durch die Initiative und die Fähigkeit widergespiegelt, neue Anwendungen und innovative Funktionen zu erstellen, die Benutzer:innen zu mehr Produktivität verhelfen.

Der IT-Betrieb

Der IT-Betrieb, für den im Deutschen häufig die englische Bezeichnung IT Operations verwendet wird, bzw. die IT-Administrator:innen sind dafür verantwortlich, dass alles optimal abläuft. Dazu müssen Hardware und Software zur Verfügung stehen und funktionieren. Zu den Hauptaufgaben von IT-Administrator:innen gehört es auch, den reibungslosen Ablauf von Datenverkehr, Infrastrukturpflege, Troubleshooting von Systemen und Datenbanken zu gewährleisten. So besteht die Rolle eines:einer klassischen IT-Betriebsspezialist:in darin, eine stabile und optimale Infrastruktur bereitzustellen. Dazu müssen Änderungen minimiert werden, sodass Benutzer:innen eine stabile und effiziente Infrastruktur zur Verfügung steht.

Widerspruch

Grundsätzlich verfolgen Entwickler:innen und der IT-Betrieb ein gemeinsames Ziel, nämlich die Produktivität der Organisation zu gewährleisten. Dennoch ist offensichtlich, dass die Rollen dieser Abteilungen in Widerspruch zueinander stehen und ihre Zusammenarbeit definitionsgemäß aufgrund der unterschiedlichen Ziele paradox ist. Das Entwicklungsteam versucht, so schnell wie möglich Software zu erstellen und zu aktualisieren, während das IT-Team Änderungen der Umgebung um jeden Preis zu verhindern sucht. In Anbetracht dieser Tatsache zielen DevOps-Praktiken darauf ab, das Hin-und-Her zwischen

den Teams zu vermeiden und die Gegensätzlichkeit durch neue Kommunikationsstandards, engere Zusammenarbeit, bessere Integration und die Automation von nebensächlichen Prozessen auszugleichen.

Lifecycle Management in der DevOps-Kultur

Modernes Lifecycle Management kann Ansätze für agile Entwicklung unterstützen, indem alle Disziplinen integriert werden, wodurch die Zusammenarbeit der Teams einer Organisation effizienter wird. Die Einführung von DevOps fördert die kontinuierliche Lieferung von Software und Updates mit häufigem Release, oft sogar mehrmals am Tag, während ganz neue Releases nur im Abstand von einigen Monaten oder ein Mal im Jahr herausgegeben werden. Das Lifecycle Management in einer DevOps-Kultur stellt auch den Rahmen für die Softwareentwicklung dar und erleichtert ein kontinuierliches Softwaremanagement.

Als Best Practices gelten die Erstellung eines knappen, fertigen Plans und der Anforderungen, durch die aus einer Idee eine Anwendung wird. Bei der Entwicklung von Software mit DevOps-Prinzipien sollte der gesamte Lebenszyklus der Anwendung bedacht werden. Dazu gehören die Softwarewartung und Softwareupdates, die Außerbetriebnahme und der Ersatz. Durch die Kombination dieser Faktoren wird beim DevOps-Lifecycle Management die Bereitstellung beschleunigt, die Workflow-Transparenz verbessert, zudem werden hochwertigere Produkte geliefert und die Entwicklerzufriedenheit gefördert (Golden, 2014). Durch die konsequente Anwendung des Application Lifecycle Managements anhand von **kontinuierlicher Integration** und **kontinuierlicher Lieferung** - zwei Begriffen, die im folgenden Abschnitt behandelt werden - werden Unzulänglichkeiten in der Erstellungskette einer Lösung früh entdeckt. Die automatisierte Qualitätssicherung und die Verteilung der entsprechenden Produktartefakte tragen dazu bei, dass die Lösung nachhaltig und von hoher Qualität ist. Der Begriff DevOps vereint Entwicklung und Betrieb, woraus Vorteile und Verbesserungen für eine Organisation resultieren.

Kontinuierliche Integration

Eine Praktik, bei der alle Arbeitskopien der Entwickler:innen mehrmals am Tag in eine Hauptlinie kopiert werden.

Kontinuierliche Lieferung

Ein Ansatz, nach dem Software in kurzen Zyklen entwickelt wird, sodass sie jederzeit freigegeben werden kann.

2.2 Dev und Ops im Betriebsalltag

Entwicklung und Betrieb sollten besser zusammenarbeiten. Das Zaubwort hierfür lautet DevOps, denn mit DevOps können agile Arbeitsmethoden in den Arbeitsalltag eines Unternehmens integriert werden. Heute werden Anwendungen immer häufiger durch agile Arbeitsweisen entwickelt (AOE, o. D.). Dennoch dauert es oft Tage oder Wochen, bis diese Anwendung von den internen Qualitäts- und Betriebsprozessen in Betrieb genommen werden können, da die Bereitstellung nicht nur durch die verwendeten Werkzeuge, sondern auch von den zugrundeliegenden Vorgehensweisen aufgehalten wird. Die Automatisierung wichtiger IT-Prozesse fördert die Effektivität und Effizienz. Die dabei erzielte Transparenz und die Feedbackmöglichkeiten bringen jedoch zusätzliche Schwächen hervor. Das sollte nicht auf Kosten der Qualität gehen. Die DevOps-Kultur ist ein großer Schritt auf dem Weg, die Entwicklungs- und Betriebsteams einander näher zu bringen und deren Reaktionen auf Änderungen zu beschleunigen. Es ist ein großer Schritt, der kulturelle Änderungen in der Softwareentwicklung mit sich bringt, und auf dem Weg zur Cloud wichtig ist. Über die kontinuierliche Integration und Lieferung hinaus bekommt eine Organisation mit DevOps mehr Feedback und einen besseren Einblick in den gesamten Softwa-

reentwicklungsprozess, da alle Ergebnisse und Schritte einheitlich automatisiert und versioniert sind. Zunächst ist es hilfreich zu prüfen, wie weit eine Organisation die DevOps-Kultur bereits umsetzen konnte. Dazu betrachten wir drei einfache Fragen, die zwar nicht den gesamten Status messen, jedoch auf den Stand der Organisation hinweisen können:

1. Haben Entwickler:innen Echtzeitzugriff auf Informationen zum Troubleshooting?
2. Werden in der Produktionsumgebung dieselben Werkzeuge und Methoden wie im Entwicklungsteam verwendet?
3. Ist die Arbeitskultur und -zusammenarbeit der Entwickler:innen und des Infrastrukturteams von Partnerschaft geprägt?

Werden diese Fragen verneint, besteht im Unternehmen noch keine DevOps-Kultur. Lautet die Antwort auf eine oder zwei Fragen „Ja“, so hat die Änderung in Richtung einer DevOps-Organisation begonnen.

DevOps einführen

Bei der Einführung des DevOps-Konzepts sollten gewisse Aspekte beachtet werden. Wie aus der bisherigen Erläuterung hervorgeht, geht es bei DevOps eher um die Kultur und Vorgehensweisen als um die Organisation. Es ist in keiner Organisation einfach, die Kultur zu verändern. Genau wie bei der Einführung einer agileren Arbeitsweise, sollten die sogenannten Soft Skills auch bei einer erfolgreichen Umstellung auf DevOps im Vordergrund stehen. Es erweist sich unter Umständen als schwierig, die Barrieren zwischen zwei historisch voneinander getrennten Unternehmensbereichen, der Entwicklung und dem Betrieb, zu überwinden. Eine erfolgreiche Verbindung wird jedoch mit hochwertigeren Apps und zufriedeneren Benutzern und Benutzerinnen belohnt (Fazliu, 2018). Es gibt einige Werkzeuge und Techniken, die bei der Verknüpfung von Vorgehensweisen helfen können, so dass sie zu einem integralen Bestandteil des täglichen Arbeitsablaufs werden. Diese Werkzeuge können auch zur Umsetzung der Best Practices verwendet werden, z. B. zum Austausch von Problemlösungsdaten zwischen der Entwicklung und dem Betrieb. Softwaretools können auch zur Prüfung des Systems nicht nur in der Entwicklungsumgebung, sondern auch in der Qualitätssicherung und in der Produktion eingesetzt werden. Es gibt Softwaretools, die während der Codeausführung zur Fehlerfindung, Timeoutmessung, Geräteparameteruntersuchung usw. nützlich sind. Die gespeicherten Protokolle können anhand von Monitoringtools im IT-Betrieb eingesehen werden. Dadurch können Codierungsprobleme leichter gefunden und Bugs schneller behoben werden. Auch der Feedback-Loop läuft reibungsloser, und auf Marktansprüche und Anpassungen kann flexibler und offener reagiert werden.

Zusammenarbeit mit den Geschäftsbereichen im Unternehmen

Die Lücke, die zwischen der Entwicklung und dem IT-Betrieb klaffte, war schon immer groß. Die Lücke zwischen den Geschäftsbereichen und der Entwicklung ist noch größer (Rossberg & Olausson, 2014). Diese beiden Bereiche einer Firma können sich aus uner-sichtlichen Gründen nur schwer auf einen gemeinsamen Ansatz für die Definition von Anforderungen einigen, die für Anwendungen und Projekte notwendig sind. Die Auswir-kungen solcher Uneinigkeiten auf den Fortschritt eines Projekts können im Lauf der Zeit verheerend werden, und viele Stakeholder:innen und Benutzer:innen haben oft das

Gefühl, dass sie nicht das System bekommen haben, was sie eigentlich gebraucht hätten. Natürlich ist es leicht, sich darüber zu beschweren, aber es ist wichtig, dass versucht wird, eine Lösung zu finden. Eine der effektivsten Methoden, um die Diskrepanz zwischen Business und Entwicklung zu vermindern, ist die Verwendung von agilen Prozessen. Obwohl es im Rahmen von DevOps keine direkte Lösung für dieses Problem gibt, ist es bereits von Nutzen, dass DevOps selbst ein agiler Ansatz ist, der die Geschäftsseite und die Entwicklungsbereiche einander näher bringt.

Kommunikation

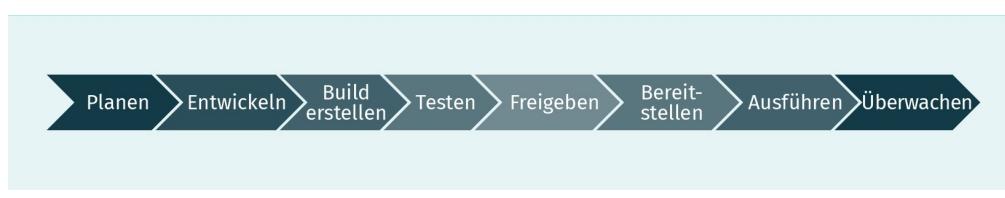
Beim Aufbau einer kooperativen DevOps-Umgebung steht zu Beginn oft die Erkenntnis, dass Abteilungen unterschiedliche Denkweisen kultiviert haben. Daher ist effektive Kommunikation besonders wichtig bei der Lösung dieses Problems. Deshalb muss auch die Kommunikation zwischen der Entwicklungs- und Betriebsabteilung verbessert werden.

Was die Unternehmenskultur angeht, so ist gegenseitiger Respekt unabdingbar. Auf der Kommunikationsebene sollten Wege verkürzt werden, denn die effektivste Kommunikation findet von Angesicht zu Angesicht statt, beispielsweise bei der gemeinsamen Arbeit zweier Mitarbeitenden an einem Whiteboard. Natürlich muss der IT-Betrieb in der Lage sein, operative Anforderungen formulieren zu können. Wenn diese bei den Entwickler:innen auf Verständnis treffen sollen, ist ein respektvoller Umgang miteinander notwendig. Es überrascht nicht, dass eine gemeinsame Sprache eine Grundvoraussetzung darstellt.

DevOps-Pipeline

Eine der wichtigsten Voraussetzungen für die Schaffung einer gemeinsamen Sprache, die das Ziel hat, die Kommunikation zu verbessern, ist die Existenz eines grundlegenden Konzepts, das von allen Stakeholder:innen, besonders dem Entwicklungs- und IT-Betrieb und den Qualitätssicherungsteams, verstanden wird. Dieses Grundkonzept kann als Automation des Übergangs von der Programmierung zum Betrieb definiert werden, so wie bereits in vielen vorherigen Schritten erläutert. Dieser Pfad von der Programmierung zum Betrieb ist das Fundament für DevOps und wird als Wertstrom oder DevOps-Pipeline bezeichnet. Die DevOps-Pipeline ist die Verbindung zwischen den Entwickler:innen und dem Betrieb, d. h. dass sie diese enger zusammenbringen kann (TheDev, 2019). Die folgende Abbildung zeigt die DevOps-Pipeline.

Abbildung 14: DevOps-Pipeline



Quelle: Kobdani (2020).

Jeder Schritt in der DevOps-Pipeline wird im Folgenden knapp erklärt:

Planen

Bevor Entwickler:innen mit dem Codieren beginnen, muss zunächst ein gründlicher Plan des gesamten Workflows vorhanden sein. Dabei spielen Produkt- und Projektmanager:innen eine wichtige Rolle. Es liegt in ihrer Verantwortung, einen Produktionsplan zur Führung des gesamten Teams durch den Prozess aufzustellen. Nach der Sammlung von Input und relevanten Informationen von den Benutzer:innen und Stakeholder:innen wird der Auftrag in einzeln aufgelistete Aufgaben unterteilt. Durch dieses Aufbrechen in kleinere, überschaubare Teile können schneller Ergebnisse erzielt, auftretende Probleme abgearbeitet und auf unerwartete Änderungen reagiert werden. DevOps-Teams arbeiten in kurzen Sprints (meist zwei Wochen), während derer einzelne Teammitglieder die ihnen zugeteilten Aufgaben bearbeiten.

Entwickeln

Während dieser Phase beginnen die Entwickler:innen mit der Codierung. Je nach Programmiersprache installieren sie die notwendige integrierte Entwicklungsumgebung, Editors und andere Softwaretools auf ihren Computern, um so effizient wie möglich arbeiten zu können. Manchmal müssen sich Entwickler:innen Codierstile und -richtlinien aneignen, sodass ihr Code einem klaren Muster folgt, damit andere Teammitglieder diesen leicht lesen und verstehen können.

Wenn Entwickler:innen so weit sind, dass sie ihren Code in das Quellcode-Repository senden können, schicken sie einen Pull Request. Der neue Code wird manuell geprüft und in die Master-Version eingeführt.

Build

Die Build-Phase ist wichtig, da hierbei Codierungsfehler zum Vorschein kommen, bevor sie durch die Pipeline geschickt werden und möglicherweise größere Probleme verursachen. Typischerweise wird durch den Pull Request ein automatisierter Prozess angestoßen, bei dem der Code in einen Build, ein bereitstellbares Paket oder eine ausführbare Datei kompiliert wird. Dies trifft natürlich nur auf Programmiersprachen zu, die kompiliert werden müssen. Zum Beispiel müssen Anwendungen in Java im Gegensatz zu Anwendungen in Python kompiliert werden. Bei einem Codefehler wird der Build abgebrochen und der:die Entwickler:in wird darüber informiert. Der ursprüngliche Pull Request schlägt dann ebenso fehl. Dieser Schritt findet jedes Mal statt, wenn ein:e Entwickler:in einen Code ins Repository schickt, so dass nur ein fehlerfreier Code in die Pipeline gelangt.

Testen

Bei Erfolg des Builds wird getestet. Das heißt, Entwickler:innen führen manuelle und automatisierte Tests durch, um die Qualität des Codes weiter zu verifizieren. In vielen Fällen wird auch ein **Akzeptanztest** durchgeführt; dabei interagieren Personen als Benutzer:innen mit der Anwendung, um zu entscheiden, ob weitere Änderungen am Code vorgenommen werden müssen, bevor er in die Produktion geht. Gleichzeitig können automatisierte Tests durchgeführt werden, um die Sicherheit der Anwendung und die Einhaltung bewährter Praktiken zu überprüfen, um nach Infrastrukturänderungen zu suchen und die

Akzeptanztest

Test zur Prüfung der in einer Spezifikation oder einem Vertrag festgelegten Anforderungen

Leistung oder die Auslastung zu testen. Das Unternehmen muss daraufhin entscheiden, was für die Anwendung wichtig ist und ob die Tests zu diesem Zeitpunkt durchgeführt werden sollten. Jedoch bietet sich diese Phase für das Testen an, denn es können neue Tests eingeführt werden, ohne den Entwicklungsfluss zu unterbrechen oder die Produktionsumgebung zu stören.

Release

Die Release-Phase, im Deutschen auch als Freigabephase bezeichnet, stellt einen Meilenstein in DevOps-Pipelines dar, an dem ein Build fertig für die Bereitstellung in der Produktionsumgebung ist. Bis dahin wurden eine Reihe manueller und automatisierter Tests an verändertem Code durchgeführt, und das Betriebsteam kann davon ausgehen, dass weitreichende Software-Fehler unwahrscheinlich sind. Je nachdem wie ausgereift die DevOps-Kultur bereits ist, kann der IT-Betrieb Builds, die es bis in diese Phase der Pipeline geschafft haben, automatisch bereitstellen. Um die neue Funktionalität auszublenden, können Entwickler:innen Funktionskennzeichnungen aktivieren, sodass Kund:innen diese erst dann sehen, wenn sie wirklich betriebsbereit sind. Alternativ kann ein Unternehmen die Kontrolle über Builds behalten, die in die Fertigung geschickt werden. Es kann täglich Release-Termine ansetzen oder bis zur Erreichung eines Meilensteins nur neue Funktionen freigeben. In der Release-Phase kann ein manueller Genehmigungsmechanismus eingeführt werden, durch den nur bestimmte Mitarbeitende die Entwicklung des Release genehmigen können. Die Technologie kann entsprechend konfiguriert werden. Es obliegt dem Unternehmen, zu entscheiden, wie es damit umgeht.

Bereitstellen

Das Programm kann in die Produktion gehen, wenn die Kompilierung den Bereitstellungszeitpunkt erreicht. Wenn am Code noch kleinere Änderungen vorgenommen werden müssen, wird eine automatische Bereitstellungsmethode verwendet. Wenn noch größere Änderungen anfallen, wird der Build in einer produktionsähnlichen Umgebung bereitgestellt, sodass die Funktion des neuen Codes geprüft werden kann. Bei wichtigen Updates wird häufig die **Blau-Grün-Bereitstellung** als Strategie verwendet.

Blau-Grün-Bereitstellung

Methode zur Installierung von Änderungen an Web-, Anwendungs- oder Datenbankservern, bei der abwechselnd Produktions- und Staging-Server verwendet werden.

Für die Blau-Grün-Bereitstellung müssen zwei gleiche Entwicklungsumgebungen bereitgestellt werden. In einer wird die aktuelle Version der Anwendung gehostet, in der anderen die modifizierte Version. Entwickler:innen sollten alle Release-Aufträge leicht an die relevanten Server weiterleiten und somit Aktualisierungen für Benutzer:innen freigeben können. Sie sollten bei Problemen auch mühelos und ohne Serviceunterbrechungen in eine alte Entwicklungsumgebung zurückkehren können.

Ausführen

Nun ist die aktuelle Version verfügbar und kann durch Kundschaft genutzt werden. Das heißt, dass das Betriebsteam für den reibungslosen Ablauf sorgt. Je nach Hosting-Server und -Diensten wird die Umgebung automatisch an die Zahl der aktiven Benutzer:innen angepasst. Das Unternehmen ermöglicht den Kund:innen, Serviceangebote zu nutzen und stellt darüber hinaus die passenden Tools zur Verfügung, mit denen Feedback gegeben

und somit die Weiterentwicklung des Produkts gewährleistet werden kann. Kund:innen wissen am besten, was sie brauchen, und Benutzer:innen sind die besten Tester, die weit mehr Zeit mit der Anwendung verbringen, als es in einer DevOps-Pipeline je möglich ist.

Überwachen

Die letzte Phase der DevOps-Pipeline ist die Überwachung des laufenden Systems. Sie baut auf dem durch Datensammlung und -analyse generierten Kund:innenfeedback. Nun ist auch Zeit für eine Selbstprüfung und die kritische Betrachtung der DevOps-Pipeline, z. B. auf Engpässe hin, die Verwirrung stiften oder die Produktivität des Entwicklungs- und Betriebsteams hindern können. Die Daten aus diesen Vorgängen werden an den oder die Produktmanager:in und das Produktionsteam übermittelt, womit die Prozessschleife geschlossen wird. Praktischerweise würde hier eine neue Schleife beginnen, allerdings ist diese Phase eine kontinuierlich fortlaufende. Es gibt also keinen Anfang oder Ende, sondern eine ständige Weiterentwicklung des Produkts über dessen Lebensdauer hinweg. Diese endet erst, wenn das Produkt nicht mehr gebraucht wird.

DevOps - Best Practices

Um die Vorteile von DevOps zu nutzen, benötigt eine Organisation saubere Implementierungspläne. Es wurde bereits erläutert, was DevOps ist, warum es notwendig ist und welche Schritte die DevOps-Pipeline umfasst. Nun können wir einen Blick auf einige bewährte Praktiken von DevOps werfen (Patel, 2020).

Konfigurationsmanagement

Das Konfigurationsmanagement ist eine wesentliche Komponente des DevOps-Prozesses. Dabei werden alle Infrastruktureinheiten und -systeme (z. B. Server, Datenbanken und andere Speichersysteme, Betriebssysteme, Netzwerk, Anwendungen und Software), die zur Installierung, der Verwaltung und der Wartung notwendig sind, automatisiert. Wird zum Beispiel eine Anwendung, die auf einer Datenbank beruht, benutzt, können die Informationen über die Verbindung zur Datenbank in einem Konfigurationsmanagementsystem gespeichert und bei Bedarf abgerufen werden. Der Vorteil des Konfigurationsmanagements liegt beispielsweise darin, dass neue Umgebungen leichter eingerichtet werden können, dass die Konfiguration der Produktionsumgebung vereinheitlicht wird und dass Zeit und Ressourcen bei der Softwareerstellung gespart werden können, anstatt diese mit dem nachstehend erläuterten Verfahren des „Infrastructure as Code“ zu verschwenden, um neue Dienste von Grund auf zu erstellen.

Release-Verwaltung

Bei der Release-Verwaltung im Rahmen einer DevOps-Kultur geht es um die Konzipierung, Zeitplanung und Verwaltung von Produktions- und Verteilungsprozessen, die zur Erstellung von Anwendungen notwendig sind. Von Anfang bis Ende des Prozesses kooperieren alle Entwickler:innen und IT-Betriebsmitglieder, wodurch weniger und kürzere Feedbackzyklen und schnellere Updates möglich sind. DevOps-Teams tragen die Verantwortung für die von ihnen angebotenen Dienste gemeinsam, für den Code und die Rufbereitschaft. Vorfälle werden sowohl während der Freigabe als auch danach schneller entdeckt und

Entwicklungszweig
Eine Kopie eines Haupt-Repository des Versions-verwaltungssystems.

bearbeitet, denn Softwareentwickler:innen und IT-Fachleute stehen während des gesamten Auslieferungszyklus zur Verfügung und leisten auch im Anschluss noch Bereitschaftsdienst.

Kontinuierliche Integration

Bei der heutigen Anwendungsentwicklung arbeiten mehrere Entwickler:innen gleichzeitig an verschiedenen Funktionen derselben App. Die gleichzeitige Zusammenführung von Quellcode aus allen **Entwicklungszweigen** an einem Tag, dem sogenannten „Merge Day“, kann enorm viel Zeit und Arbeit in Anspruch nehmen. Denn Änderungen, die von vielen, voneinander unabhängig arbeitenden Entwickler:innen erstellt wurden, können zu Unvereinbarkeiten führen, wenn sie gleichzeitig zusammengeführt werden. Dieses Problem kann dadurch verstärkt werden, wenn Entwickler:innen ihre eigenen lokal eingerichteten Entwicklungsumgebungen haben, anstatt gemeinsam in cloudbasierten integrierten Entwicklungsumgebungen zu arbeiten. Anhand der kontinuierlichen Integration können Codeänderungen öfter, manchmal täglich, in einen gemeinsamen Entwicklungszweig bzw. einen Hauptentwicklungszweig zusammengeführt werden. Wurden alle Änderungen zusammengeführt, werden sie durch automatisierte Anwendungsbuids und verschiedene Ebenen automatischer Tests (im Normalfall Modul- und Integrationstests) validiert. Dadurch wird gewährleistet, dass die Funktionalität nicht kompromittiert wurde. Alle Klassen und Funktionen bis zu den verschiedenen Modulen der Anwendung müssen getestet werden. Wenn ein automatisierter Test Unvereinbarkeiten zwischen dem alten und dem neuen Code feststellt, können diese bei der kontinuierlichen Integration schneller und häufiger gelöst werden (Patel, 2020).

Kontinuierliche Lieferung

Nach der Automatisierung von Builds sowie von Modul- und Integrationstests während der kontinuierlichen Integration kann die kontinuierliche Lieferung den validierten Code ins Repository liefern. Um eine effiziente, kontinuierliche Lieferung zu gewährleisten, muss bereits eine kontinuierliche Integration in der Entwicklungspipeline bestehen. Das Ziel der kontinuierlichen Lieferung ist es, dass die Codebasis jederzeit in einer Produktionsumgebung zur Verfügung gestellt werden kann. Bei der kontinuierlichen Lieferung werden in jeder Phase – vom Zusammenführen von Codeänderungen bis zur Lieferung produktionsgerechter Builds – automatisierte Tests und Codefreigaben durchgeführt. Am Ende dieses Prozesses kann das Betriebsteam die Anwendung schnell und einfach in der Produktionsumgebung bereitstellen (Patel, 2020).

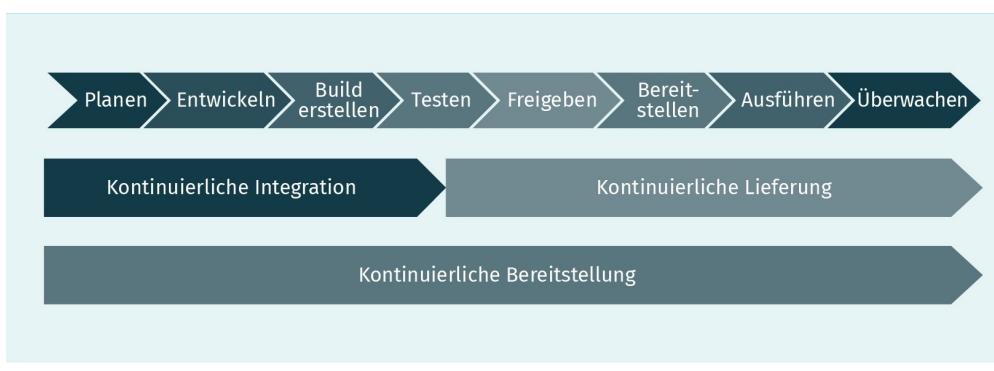
Kontinuierliche Software-Verteilung

Die kontinuierliche Software-Verteilung kann als Erweiterung der kontinuierlichen Lieferung betrachtet werden, bei der produktionsgerechte Builds automatisch ins Code-Repository geliefert werden. Bei der kontinuierlichen Software-Verteilung wird auch die Lieferung einer Anwendung in die Produktionsphase automatisiert. Da der Produktionsphase in der Pipeline kein manueller Check vorgeschaltet ist, muss automatisiertes Testen bei der kontinuierlichen Software-Verteilung immer gut durchdacht sein. In der Praxis bedeutet die kontinuierliche Software-Verteilung, dass Änderungen an der App innerhalb weniger Minuten nach Erstellung in Betrieb genommen werden können. Sie müssen dazu nur

die automatisierten Tests bestehen (Patel, 2020). Dadurch wird die kontinuierliche Integration von Benutzer:innenfeedback viel einfacher. Alle zusammenhängenden Praktiken der kontinuierlichen Integration und Bereitstellung, was oft mit CI/CD (vom englischen „continuous integration/continuous delivery“) abgekürzt wird, reduzieren Implementierungsrisiken, da Änderungen in Teilen statt in Gänze freigegeben werden. Es muss dafür jedoch zuvor einiges, z. B. an automatisierten Tests für die verschiedenen Testphasen der CI/CD-Pipeline, investiert werden.

Das folgende DevOps-CI/CD-Diagramm zeigt einen einfachen Vergleich zwischen kontinuierlicher Integration, Lieferung und Bereitstellung.

Abbildung 15: DevOps mit kontinuierlicher Integration und Bereitstellung



Quelle: Kobdani (2020).

Infrastructure as Code

Als „Infrastructure as Code“, im Deutschen auch „programmierbare Infrastruktur“, wird die Definition aller Software-Runtime-Umgebungen, Netzwerkeinstellungen und -parameter bezeichnet, die auf Anfrage in einfachem Textformat im Code-Repository gespeichert bzw. modifiziert werden kann (Null, o. D.). Diese Textdateien werden als Manifeste bezeichnet und automatisch durch die Infrastruktursoftware zur Erstellung von Server-, Test-, Staging- und Entwicklungsumgebungen bereitgestellt und konfiguriert. Besonders nennenswert ist, dass alle Vorgänge anhand des Code-Repository nachverfolgt werden können. Dadurch ist das jahrzehntelange Problem, dass der Code nur in der Testumgebung, nicht aber in der Produktionsumgebung funktioniert hat („auf meinem Computer funktioniert es“), endlich hinfällig. „Infrastructure as Code“ gewährleistet Beständigkeit, da alle Umgebungen automatisch bereitgestellt und konfiguriert werden. Auf diese Weise wird menschlichen Fehlern vorgebeugt, wodurch wiederum die Softwareentwicklung und der Infrastrukturbetrieb beschleunigt und vereinfacht werden.

Testautomatisierung

Durch automatisiertes Testen jeder Codebasis können Entwickler:innen mehr Tests durchführen, zudem steigt die Testgeschwindigkeit und es wird Zeit bei der manuellen Qualitätssicherung gespart. Dadurch kann die Fehlerfindung und -behebung früher stattfinden

und die Gesamtqualität von Anwendungen steigt. Einige Tools für die Testautomatisierung, wie Selenium, Appium und Junit, können mit DevOps-Tools kombiniert werden (Verona, 2016).

Kontinuierliches Monitoring

Kontinuierliches Monitoring bedeutet den Einsatz verschiedener Tools, Dashboards und Benachrichtigungen zur Überwachung aller Systeme und der Infrastruktur. Dazu gehört auch die Prüfung verschiedener, die Software beeinflussender Werte, wie Systemleistung, Testanzahl, Erfolgs-/Misserfolgsraten, Implementierungsstatus, Fehlerprotokolle und umfassende Berichtsformatinformationen in Grafik- oder Tabellenform. Auf dem Markt sind einige Tools vorhanden, mit denen man sich einen Werkzeugkasten für das DevOps-Monitoring zusammenstellen kann. Dazu gehören z. B. Prometheus, Grafana, Nagios, Appdynamics, NewRelic, Splunk oder Logstash (Elastic, o. D.).

Site Reliability Engineering

Site Reliability Engineering (SRE) und DevOps sind moderne Disziplinen, die sich stark überschneiden. SRE wird von manchen sogar als Konkurrenzpraktik zu DevOps bezeichnet (Vargo & Fong-Jones, 2018). SRE besteht aus Softwareentwicklung und -produktion. Ein SRE-Team verbringt die Hälfte der Zeit mit der Entwicklung und die andere Hälfte mit dem Betrieb. Ähnlich wie bei der DevOps-Kultur liegt die Stärke von SRE in der Kommunikation und in der Informationsweiterleitung zwischen der Entwicklungs- und der Betriebsabteilung, allerdings mit zusätzlichen Zielen (Wikipedia, 2020b).

2.3 Auswirkungen auf die Team- und Entwicklungsstruktur

Silos

Bei diesem Management-System kommt es zu einer Struktur, bei der sich Teams auf die eigenen Ziele konzentrieren, anstatt auf Unternehmensziele ausgerichtet zu sein.

Eines der Hauptaugenmerke liegt bei DevOps-Organisationen in der reibungslosen Zusammenarbeit zwischen den Entwicklungs- und Betriebsteams. DevOps wurde mit dem Ziel entwickelt, **Silos** zu verhindern, so dass Anwendungen gemeinsam und somit schneller entwickelt, getestet und bereitgestellt werden können (DevOps, 2015). DevOps ist jedoch wesentlich mehr als eine Theorie oder eine ansprechende Abkürzung, denn die Systemkomponenten gehen weit darüber hinaus.

Die Antwort auf die Frage, was die Idealstruktur eines DevOps-Teams ist, ist vielschichtig. Eine bestehende Struktur kann nur dann in der DevOps-Kultur funktionieren, wenn die Entwicklung und der Betrieb so zusammenarbeiten, dass Geschäftsziele erreicht oder übertroffen werden können. Dieses Ziel nimmt in jedem Unternehmen eine andere Form an. Das machen wir uns bei der Analyse der verschiedenen Modelle zu Nutzen. Was am besten zu einem Team passt, lässt sich durch eine Analyse der Vor- und Nachteile der jeweiligen Modelle ermitteln. Bei der Teamstruktur spielen verschiedene Faktoren eine Rolle.

- **Bestehende Silos:** Arbeiten Teams allein?

- **Führung:** Wer übernimmt die Führung des Teams und welche Branchenerfahrung bringen diese leitenden Personen mit? Haben Entwicklung und Betrieb dieselben Prioritäten oder werden sie von der Erfahrung des jeweiligen Leiters oder der Leiterin beeinflusst?
- **IT-Betrieb:** Sind die Aktivitäten an den Prioritäten des Unternehmens ausgerichtet oder geht es nur darum, Server einzurichten und das Entwicklungsteam bei der Arbeit zu unterstützen?
- **Wissenslücken:** Sind die Fähigkeiten und Talente zur Umstellung auf DevOps im Unternehmen vorhanden?
- **Architektur:** Erfüllen die Prinzipien für das Architekturdesign die Anforderungen zur Bildung von DevOps-Teams?

Teamgröße

Amazon hat mit der Idee der Zwei-Pizza-Teams in der DevOps-Welt Aufsehen erregt. Danach sei das leistungsstärkste Team klein genug, um von zwei Pizzen satt zu werden (Buchanan, 2019). Kleinere Teams führen zu lockeren Strukturen und Microservices, die von leistungsfähigen DevOps-Organisationen bevorzugt werden. Eine Reorganisation in kleinere Teams ist jedoch leichter gesagt als getan und kann eine Organisation überfordern. Ein Ansatz, mit dem der Prozess begonnen werden könnte, ist die Bildung einer Portfolioorganisation. Zu Portfoliotools gehören Entwickler:innen, IT-Fachleute, Tester:innen, klassische Projektmanager:innen usw. Sie können funktionsbezogen oder rollenbezogen arbeiten, sollten jedoch alle das Ziel verfolgen, die Fähigkeiten der Firma zu verbessern und schneller einen Wert an die Kund:innen liefern zu können. Eine Firma könnte einen Vizepräsidenten oder eine Technologiedirektorin einstellen, der oder die eine Portfoliostruktur für die Organisation entwirft. Nachdem der Entwurf steht, leitet der Vizepräsident einige kleinere, von Teamleiter:innen geführte DevOps-Teams, die an den Firmenzielen ausgerichtet sind.

Führung

Angenommen, in einem Unternehmen existiert ein hoher Grad an Instabilität und es besteht ein großer Bedarf, die Kommunikation zur Stärkung der DevOps-Kultur zu verbessern. Dann muss unbedingt zunächst ermittelt werden, wie am besten vorgegangen wird. Der Führungsstil kann dabei helfen, die Schwierigkeiten, die bei der Implementierung von DevOps auftreten, anzugehen. Veränderungen in einer Organisation lassen sich oft schwer umsetzen: Der Ansatz muss unternehmensweit erfolgen und mehrere Abteilungen müssen sich auf einen Implementierungsansatz einigen. Änderungen innerhalb einer Organisation sind nie einfach, sie erweisen sich jedoch besonders für diejenigen als schwierig, die von vornherein nicht gut miteinander arbeiten können. Bedeutende Ursachen für Misserfolg sind

- Veränderungsresistenz,
- geringe Bereitschaft zu Veränderungen und
- schwaches Mitarbeiterengagement

Als „transformationale Führung“ wird ein Führungsstil bezeichnet, bei dem Mitarbeitende gefördert, ermächtigt und motiviert werden, Änderungen im Sinne des Organisationswachstums durchzuführen und somit den möglichen Erfolg des Unternehmens mitzustalten (Wikipedia, 2020). Ein derartiger Führungsstil kann, je nach Reaktion der Teammitglieder auf die Änderungen hin zur DevOps-Kultur, erfolgversprechend sein.

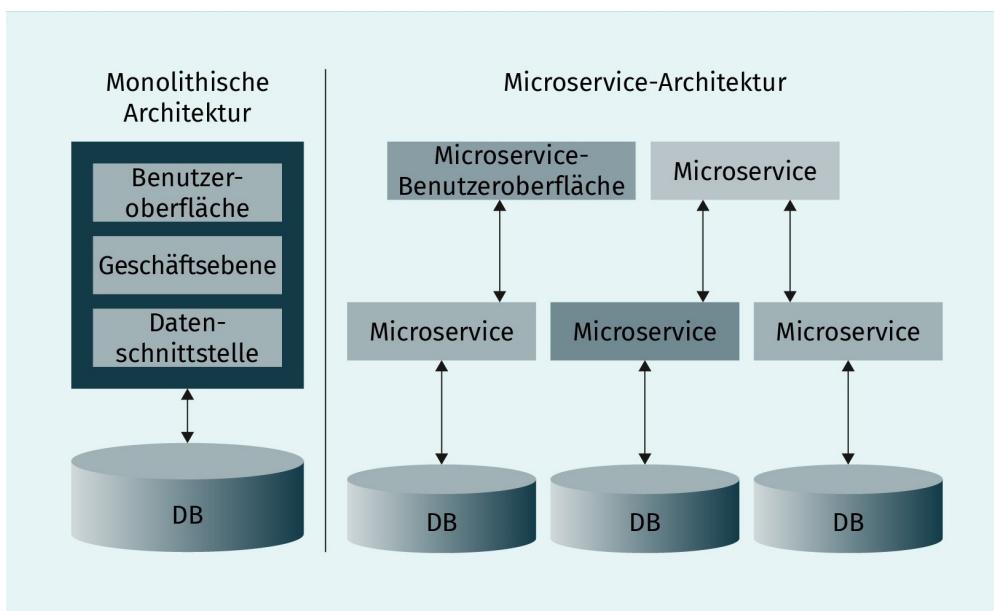
Die richtige Mischung

Auf dem Weg zur Formung von DevOps-Teams sollten als erstes die Qualifikationsdefizite analysiert werden. Dafür werden zunächst die Rollen und Fähigkeiten bestimmt, die die Mitarbeitenden zur Umsetzung der Ziele des Teams benötigen. In diesem Sinne muss eruiert werden, welche Teammitglieder derzeit welche Stellen innehaben, wie viele zusätzliche Mitarbeitende eingestellt werden müssen und welche Fähigkeiten diese mitbringen müssen, um das Defizit zu beheben. Dabei spielen nicht nur die fachlichen Qualifikationen oder Aufgaben eine Rolle. Das Management sollte auch die zusätzlich notwendigen interpersonellen Fähigkeiten und persönlichen Eigenschaften ermitteln, die zu einem ausgeglichenen Team führen.

Microservice-Architektur

Apps können auf verschiedene Arten entwickelt werden. Anstatt beispielsweise ein Projekt als Ganzes, also monolithisch, zu implementieren, kann sich die Verteilung der Aufgaben in kleinere Pakete (Microservices) als sinnvoll erweisen. Dieses Konzept passt nicht zur Entwicklung von Anwendungen, es spielt auch bei der Planung von agilem Projektmanagement und agiler Projektausführung eine wichtige Rolle. Die verteilte Architektur von Microservices bedeutet auch, dass sich Entwickler:innen mit der Ausführung der Anwendung beschäftigen müssen, so dass Fehlern bei der Verbindung und Synchronisierung der einzelnen Dienste vorgebeugt werden kann. Dabei kann DevOps äußerst hilfreich sein. Microservices und DevOps sind wichtige Entwicklungen, die die Firmenkultur mitbestimmen und die Organisation belastungs- und leistungsfähiger machen. Man kann sagen, dass die Grundlage für hervorragende Microservices eine ausgezeichnete DevOps-Kultur ist. Die Idee der Microservices entstammt einer Reihe von gängigen Konzepten der DevOps-Kultur vieler erfolgreicher Firmen. Diese Firmen arbeiteten anfangs häufig mit monolithischen Anwendungen, die schnell in kleinere Dienste unterteilt und mit anderen Diensten anhand von REST-APIs und anderen Kommunikationsprotokollen weitergeleitet wurden. Folgendes Diagramm zeigt den Vergleich zwischen monolithischer und Microservice-Architektur (Anji, 2020).

Abbildung 16: Monolithische Architektur im Vergleich zur Microservice-Architektur



Quelle: Kobdani (2020).

Microservice-Architekturen werden oft von IT-Abteilungen eingesetzt, um besser auf die Anforderungen von Organisationseinheiten reagieren und Anwendungen schneller bereitstellen zu können. Dabei kommt der engen Zusammenarbeit von Entwicklungs- und Betriebsteams, also der Implementierung von DevOps, eine bedeutende Rolle zu. Ein weiterer Vorteil des Microservice-Ansatzes ist es, dass kleinere Teams einzelne Dienste schneller erstellen, validieren und freigeben können, wodurch die Einführung von DevOps ebenfalls vereinfacht wird. Microservices und DevOps sind also sehr komplementär.

Der gemeinsame Einsatz beider Ansätze gepaart mit Containern erleichtert die Bereitstellung einer skalierbareren und produktiveren Infrastruktur und bietet Anwendungen, die die Infrastruktur optimal nutzen können, und führt zu Prozessen, anhand derer diese Anwendungen schnell und mit hoher Qualität entworfen und implementiert werden können.

Twelve-Factor-App

Die Twelve-Factor-App, im Deutschen auch als Zwölf-Faktoren-App bezeichnet, ist eine Methodik bzw. eine Prinzipiensammlung zur Erstellung skalierbarer und resilenter Unternehmensanwendungen (Wiggins, 2017). Sie legt allgemeine Prinzipien und Empfehlungen für die Entwicklung robuster Anwendungen fest. Heutzutage wird die Methodik häufig angewandt, da sie zur Erstellung von Microservices verwendet werden kann. Die Twelve-Factor-App ist technologieagnostisch und vollständig kompatibel mit Microservices im Rahmen von DevOps.

Die App folgt die nachstehend ausgeführten Prinzipien.

Unterstützender Dienst
Ein von der App über das Netz in Anspruch genommener Dienst und Teil der Aktivitäten in einem Vorgang.

Port
Ein Kommunikationsendpunkt in einem Computernetz, der Anwendungen bei der Kommunikation miteinander über das Internet unterstützt.

Prozessmodell
Ein Modell aus einer Sequenz von Phasen und Aufgaben, die den gesamten Systemlebenszyklus abdecken.

Ordnungsgemäßes Herunterfahren
Die Fähigkeit eines Betriebssystems, alle Prozesse sicher abzuschalten und Verbindungen sicher abzuschließen, wenn im laufenden System ein Abschaltesignal eingeht.

1. **Codebasis:** Die Entwicklung findet mit einer einzigen Codebasis statt, die durch ein Versionsverwaltungssystem nachverfolgt wird. Es sollte nur ein Repository pro Anwendung zum Einsatz kommen, um die CI/CD-Pipeline so einfach wie möglich zu gestalten.
2. **Abhängigkeiten:** Abhängigkeiten sollten explizit deklariert und getrennt werden. Konfigurationsmanagementsysteme, wie Chef (Chef, o. D.) oder Ansible (Red Hat Ansible, o. D.-a), können verwendet werden, um Abhängigkeiten auf Systemebene hinzuzufügen.
3. **Konfiguration:** Die Konfiguration sollte in der Umgebung gespeichert werden. Es erleichtert die Erstellung einer CI/CD-Pipeline und macht den Vorgang flexibler.
4. **Unterstützende Dienste:** Unterstützende Dienste werden wie angeschlossene Ressourcen behandelt. Dadurch können Ressourcenanbieter dynamisch ohne Auswirkungen auf das System ausgetauscht werden, wodurch die CI/CD-Pipeline robuster wird.
5. **Drei Phasen:** Build, Release und Ausführung. Die Build- und Ausführungsphasen sollten streng getrennt werden; zur Automatisierung der Build- und Bereitstellungsprozesse können CI/CD-Werkzeuge verwendet werden.
6. **Prozesse:** Die Anwendung sollte in einem oder mehreren zustandslosen Prozessen ausgeführt werden. Dadurch wird die Anwendung besonders skalierbar ohne Auswirkungen auf das System. Mögliche Konflikte zwischen der Entwicklung und dem Betrieb können so minimiert werden.
7. **Bindung an Ports:** Dienste werden anhand von Portbindung exportiert. Dabei geht es eher um die Erstellung eigenständiger Anwendungen als um die Bereitstellung auf einem externen Anwendungsserver. Je unabhängiger die CI/CD-Pipeline von externen Anwendungsservern ist, desto robuster ist sie.
8. **Nebenläufigkeit:** Aufskalierung erfolgt anhand eines **Prozessmodells**, wodurch horizontales Skalieren im Gegensatz zum vertikalen Skalieren bevorzugt wird.
9. **Einweggebrauch:** Anwendungen werden durch geringe Startzeit und **ordnungsgemäßes Herunterfahren** robuster. Wenn ein System strapazierfähiger ist, wird auch die CI/CD-Pipeline belastbarer.
10. **Dev-Prod-Vergleichbarkeit:** Die Entwicklungs-, Staging- und Produktionsphasen sollten so ähnlich wie möglich gestaltet werden. Dadurch wird das Risiko von Programmfehlern in einer Umgebung reduziert, wodurch Inkonsistenzen vermindert werden. Konflikte können in der Phase zwischen Entwicklung und Produktion auftreten.
11. **Protokolle:** Protokolle können als Ereignisstrom betrachtet werden. Sie sind bei der Problembehandlung in der Produktionsumgebung unerlässlich, da sie Einblick in die Aktionen im ausgeführten Programm geben und ein Kommunikationstool zwischen Entwicklung und Betrieb sind.
12. **Admin-Prozesse:** Administrative Prozesse sollten als inklusive Prozesse ausgeführt werden. Die Modularität der benötigten Prozesse wird dadurch verbessert, was wiederum die Modularität der CI/CD-Pipelinekomponenten erhöht.

2.4 Aufbau einer DevOps-Infrastruktur

In diesem Abschnitt befassen wir uns mit dem Aufbau einer DevOps-Infrastruktur. Obwohl eine umfassende Diskussion dieses Themas hier nicht möglich ist, soll zumindest der Grundstein gelegt werden. Wie zuvor gesagt, ist DevOps eine Kultur, die auf die Förderung

von Zusammenarbeit und Kommunikation zwischen Entwicklungs- und Betriebsteams abgerichtet ist. Die Einführung der CI/CD-Pipeline wird oft zur technischen Implementierung des DevOps-Ansatzes empfohlen. Die verschiedenen Schritte der CI/CD-Pipeline stellen die verschiedenen Unteraufgaben, die in Pipelinephasen unterteilt sind, dar. Diese Schritte sind generell denen der DevOps-Pipeline ähnlich. Sie können sich jedoch je nach gewählter und eingesetzter Technologie auch von ihnen unterscheiden. Da es immer mehr CI/CD-Werkzeuge auf dem Markt gibt, ist es nicht leicht, die richtigen auszusuchen. Allerdings gibt es relativ wenige, die schon seit Jahren auf dem Markt und in weitverbreiteten Einsatz sind. Ein Tool, das näher besprochen werden sollte, ist Jenkins.

CI/CD und Jenkins

Jenkins ist ein führender Open-Source-Automatisierungsserver, der Hunderte von Plugins zur Unterstützung von Build-, Bereitstellungs- und Automatisierungsprozessen jeglicher Projekte anbietet. Jenkins ist eine Java-Anwendung und wird in Web-**Containern** ausgeführt. Es kann mit vielen verschiedenen **Versionsverwaltungssystemen** verwendet werden und hat Plugins für verschiedene Technologien und Bereitstellungsszenarien (Jenkins, o. D.-a). Eine der bekanntesten Kombinationen für eine robuste Infrastruktur mit einer CI/CD-Toolskette besteht aus Git, Jenkins, Docker und Kubernetes.

Git ist ein Open-Source-Versionsverwaltungssystem. Mit Git kann die Versionierung von Projekten und Dateien, die sich durch die Mitarbeit vieler ändern, vorgenommen werden (Git, o. D.) Docker wird zur Isolierung von Anwendungen anhand von Container-Virtualisierung verwendet. Container sind für die unabhängige Bereitstellung und Ausführung von Microservice-Apps ideal, da mehrere Teile einer Anwendung in Microservices auf derselben Hardware unabhängig voneinander ausgeführt werden können. Gleichzeitig können die einzelnen Komponenten und Lebenszyklen besser gesteuert werden (Docker, o. D.-a). Docker vereinfacht die Bereitstellung von Anwendungen, da Container mit allen notwendigen Paketen leicht transportiert und als Dateien installiert werden können. Kubernetes, auch unter der Abkürzung k8s bekannt, ist eine Open-Source-Plattform, die die Automatisierung und Orchestrierung des Betriebs von Linux-Containern ermöglicht (Kubernetes, o. D.). Das folgende Diagramm zeigt das Setup mit Git, Jenkins, Docker und Kubernetes.

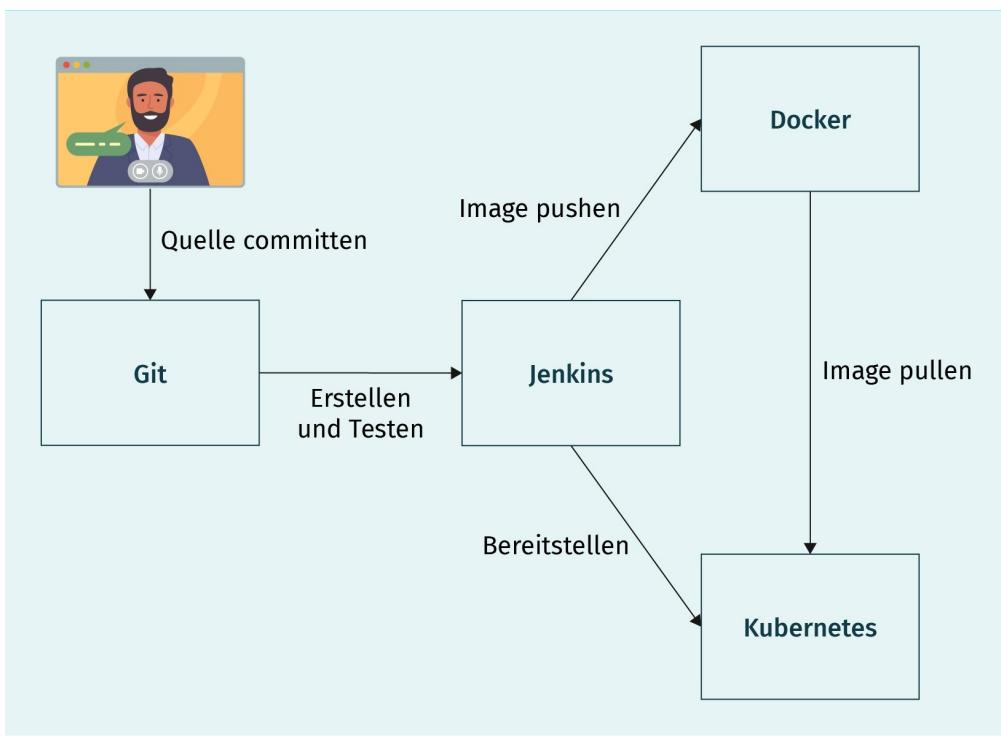
Container

Eine Standardsoftwareeinheit, in der Code und alle Abhängigkeiten zusammen gespeichert werden, sodass die Anwendung schnell und zuverlässig in verschiedenen Umgebungen ausgeführt werden kann.

Versionsverwaltungssystem

Ein System, das Änderungen an einer Datei oder einem Dateiensatz speichert, sodass bestimmte Versionen später wieder abgerufen werden können.

Abbildung 17: Kontinuierliche Integration und Bereitstellung anhand von Git, Jenkins, Docker und Kubernetes



Quelle: Kobdani (2020).

Mit dem oben dargestellten Setup werden folgende automatische Schritte durchgeführt:

1. Codeänderungen werden in Git (Git, o. D.) oder einem anderen Versionsverwaltungssystem committet.
2. Jeder Commit in GitHub löst automatisch einen Build in Jenkins aus. Jenkins verwendet beispielsweise Maven (Apache Maven Project, o. D.; Jenkins, o. D.-b), um den Java-Code eines Projekts zu kompilieren, Modultests auszuführen und weitere Prüfungen, wie z. B. Code Coverage, Codequalität usw., vorzunehmen.
3. Nach der erfolgreichen Kompilierung und Kontrolle des Codes erstellt Jenkins ein neues Docker-Image und fügt es durch einen Push in die Docker-Registrierung ein.
4. Jenkins meldet Kubernetes, dass ein neues Image zur Bereitstellung verfügbar ist.
5. Kubernetes führt einen Pull des neuen Docker-Images aus der Docker-Registrierung durch und stellt es bereit.

Zum besseren Verständnis des obigen Ablaufs müssen wir uns die Docker-Technologie näher ansehen.

Docker

Docker ermöglicht es, Anwendungen und die zugehörigen Abhängigkeiten (z. B. Laufzeit-Frameworks und Bibliotheken) in ein Image zu packen und auf jeder beliebigen Maschine auszuführen. Vereinfacht gesagt enthält ein Image die Elemente, die zur Anwendungsaus-

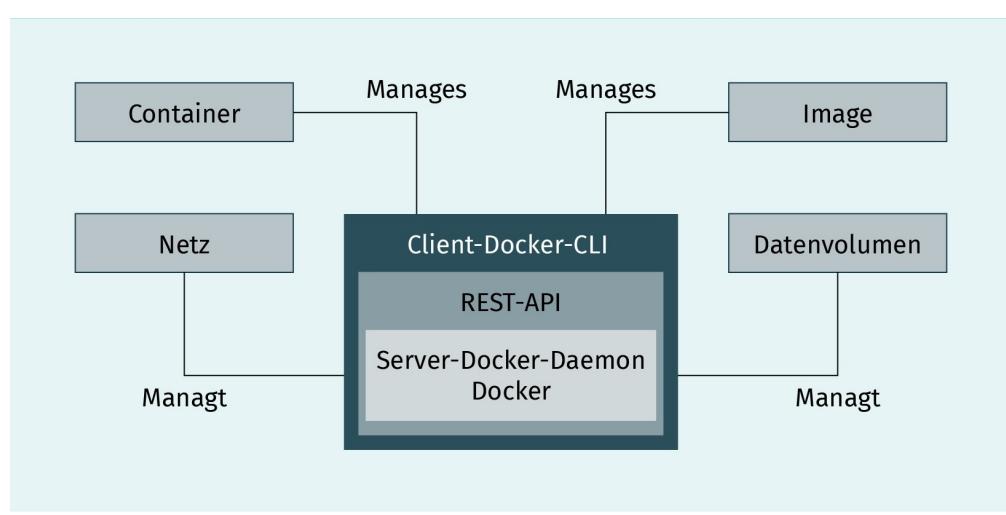
führung notwendig sind. Dazu muss die Docker-Engine nur auf dem Computer installiert sein. Docker liest dann das Image und erstellt einen Container, der auf allen Plattformen (Smartphone, Laptop oder in der Cloud) läuft. Es gibt viele Gründe, warum Docker beliebt ist (Tozzi, 2017). Zum einen wird durch Docker die bestehende Linux-Containerfunktionalität (z. B. durch die Versionsverwaltung von Images und Containern) erweitert. Zudem können Docker-Images sehr leicht beschrieben, gebaut und zwischen den Systemen transportiert werden. Sie eignen sich im Vergleich zu virtuellen Maschinen auch dazu, hochvertrauliche Ressourcen zu verwalten. Es gibt viele Unterschiede zwischen virtuellen Maschinen und Containern. Der größte liegt in der Möglichkeit, Betriebssysteme so in einem Container zu virtualisieren, dass mehrere Workloads auf einer einzigen Instanz eines Betriebssystems ausgeführt werden können. Auf virtuellen Maschinen laufen Betriebssysteminstanzen auf virtualisierter Hardware. Docker ist Open Source, verfügbar für Windows, Linux und Mac OS und hat eine große Community mit Lernprogrammen und ausgezeichneter Dokumentation.

Folgende Docker-spezifische Ausdrücke sollten Sie kennen (Docker, o. D.-b):

- **Dockerfile:** Eine Textdatei mit Befehlen zur Erstellung eines Images.
- **Image:** Ein Docker-Image besteht aus Elementen, wie Code, Config-Dateien, Umgebungsvariablen, Bibliotheken und Runtime-System, die zur Ausführung einer Anwendung als Container notwendig sind.
- **Container:** Eine Standardeinheit, die spontan erstellt werden kann, um eine bestimmte Anwendung oder eine Umgebung bereitzustellen.
- **Docker-Registrierung:** Ein Dienst für Docker-Images und -Repositories.
- **Docker-Daemon:** Ein Server, hier ein langes Serverprogramm, das als Daemon-Prozess bezeichnet wird. Ein Daemon ist ein Linux- oder UNIX-Programm, das im Hintergrund abläuft und daher auch als Hintergrundprogramm bezeichnet wird.
- **Docker-REST-API:** Schnittstellen, die Programme zur Kommunikation mit dem Daemon und zur Übermittlung von Anweisungen verwenden können.
- **Docker-Client:** Der Client hat eine Befehlszeilenschnittstelle, mit der die Container der Docker-Engine erstellt, ausgeführt und gestoppt werden können. Primär ermöglicht der Docker-Client es, Images aus der Registrierung zu pullen und auf dem Docker-Host auszuführen.
- **Docker-Engine:** Eine Client-Server-Anwendung aus Docker-Daemon, der Docker-REST-API und dem Docker-Client. Die Schnittstelle zwischen den Host-Ressourcen und den ausgeführten Containern. Docker-Container können auf allen Systemen, auf denen eine Docker-Engine installiert ist, ausgeführt werden.
- **Docker-Netz:** Docker unterstützt anhand von Netztreibern auch Netzcontainer.
- **Docker-Volume:** Der bevorzugte Mechanismus für persistente, durch Docker-Container generierte Daten, die dann auch von den Containern genutzt werden.

Das folgende Diagramm zeigt die Hauptkomponenten des Docker-Systems.

Abbildung 18: Docker-Komponenten



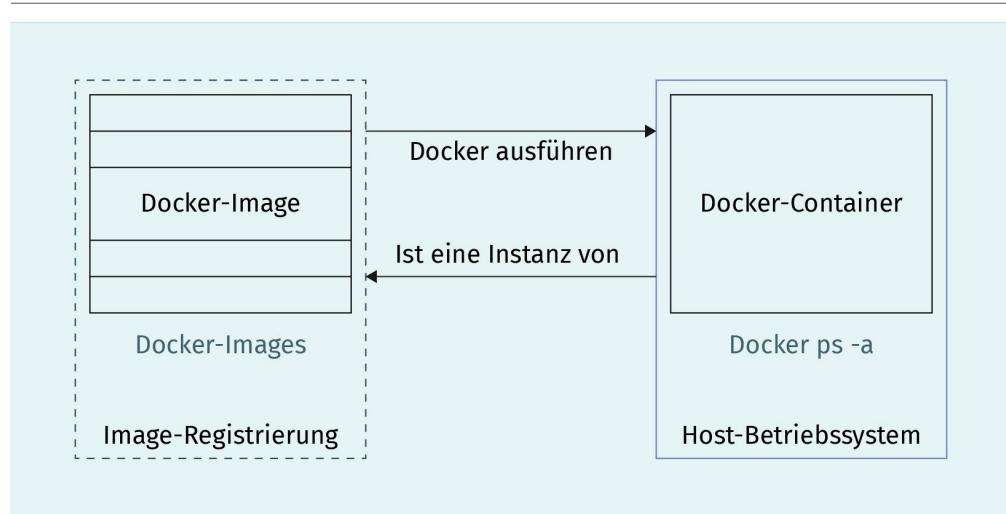
Quelle: Kobdani (2020).

Ein Container wird durch ein Image gestartet (Container bestehen aus Images). Ein Image ist ein auszuführendes Paket, das alles, was zur Ausführung nötig ist, enthält, und zwar

- den Code,
- ein Runtime-System,
- Bibliotheken,
- Umgebungsvariablen und
- Konfigurationsdateien.

Das folgende Diagramm zeigt Docker-Images und -Container und ihre Beziehung zueinander.

Abbildung 19: Docker-Image im Vergleich zu Docker-Container



Quelle: Kobdani (2020).

Die folgende Dockerfile-Datei ist ein einfaches Beispiel, wie ein Image anhand eines Skripts in Python erstellt werden kann.

Code

```
FROM python:3
ADD my-script.py /
CMD [ "python", "./my-script.py" ]
```

In der obigen Dockerfile-Datei weist FROM auf das Image, das als Grundlage des Python-Skript-Image notwendig ist.

- Das Argument `python:3` ist der Name und das Tag für das grundlegende Image, wobei das Tag nach „`:`“ kommt. Der Name des Image ist also `python` und das Tag ist `3`.
- Durch ADD wird das Skript dem Image zugefügt.
- Und CMD löst aus, dass Docker einen Befehl ausführt, wenn das Image geladen ist.

Um ein Image aus der obigen Dockerfile-Datei zu erstellen (der Name „Dockerfile“ ist in der Form erforderlich), kann der folgende Befehl im Verzeichnis, in der die Dockerfile-Datei gespeichert ist, verwendet werden. Dafür existiert „`.`“ am Ende des Befehls (auch die Anwendung `my-script.py` muss ins Verzeichnis gestellt werden):

```
$ docker build -t my-image .
```

Nach der Image-Erstellung kann ein Container mit folgendem Befehl ausgeführt werden:

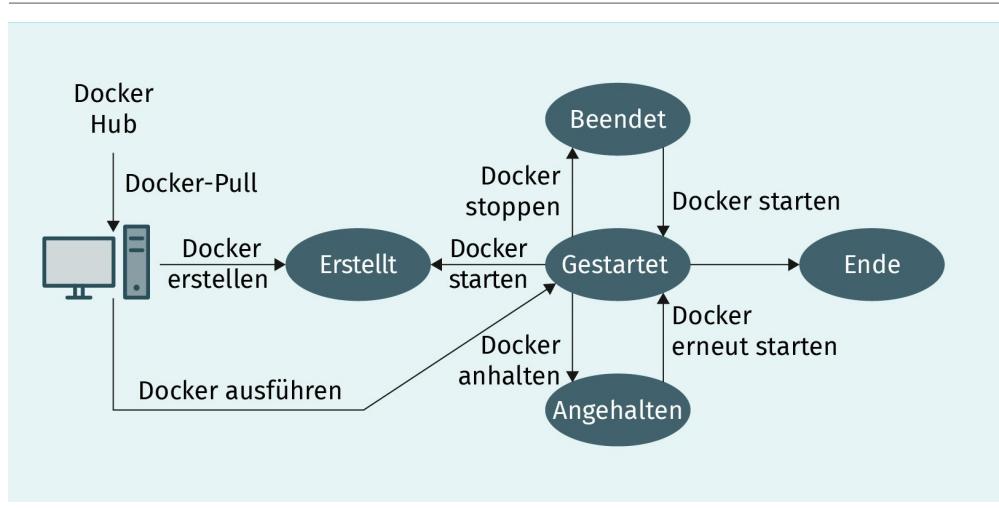
```
$ docker run my-image
```

In obigem Befehl steht `my-image` für den Namen des Images, das wir zuvor erstellt haben und das im lokalen Docker-Repository existiert. Eine interessante Option des `docker run`-Befehls ist die `-it`-Flag, eine Kombination aus `-i` und `-t`-Flags, die dazu verwendet wird, den Container zu öffnen. Dazu wird ein neuer Shell-Prozess wie folgt verwendet:

```
$ docker run -it my-image bash
```

Das nachstehende Diagramm illustriert den Lebenszyklus eines Docker-Containers und die entsprechenden Befehle.

Abbildung 20: Übersicht des Docker-Lebenszyklus



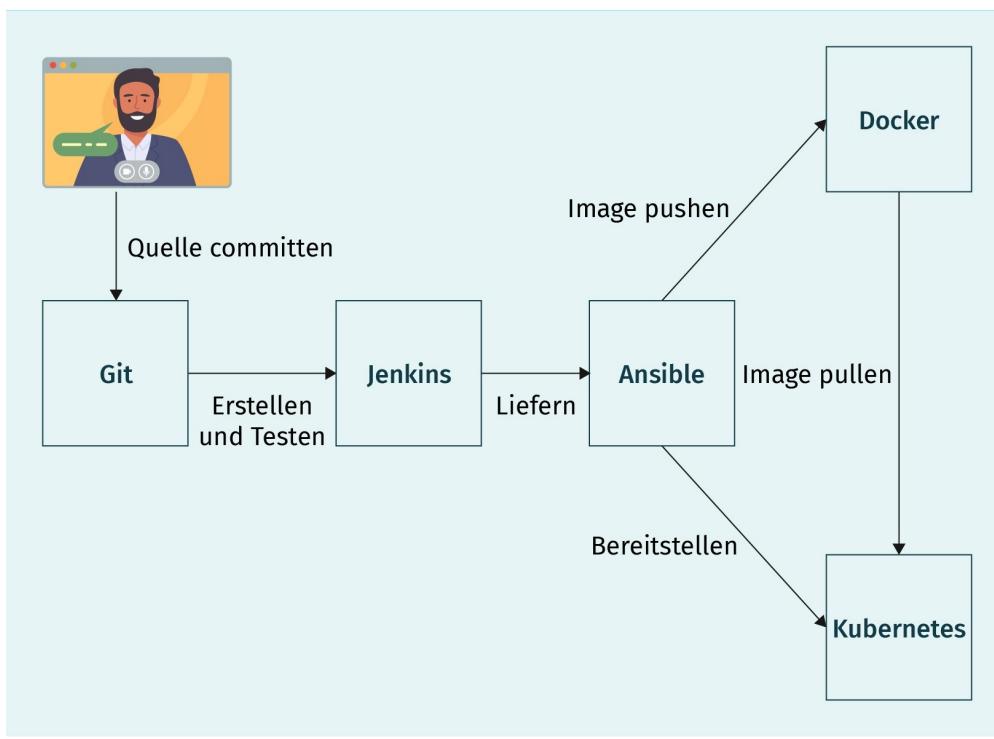
Quelle: Kobdani (2020).

Docker Hub ist ein Speicherplatz für Docker-Images für den öffentlichen Zugriff und ermöglicht es Entwickler:innen, schnell neue und zusammengesetzte Anwendungen zu erstellen. Mithilfe der Push- und Pull-Befehle können Images von der Docker Hub ausgeladen sowie auf sie zurückgeschrieben werden (Docker, o. D.-c).

Ansible

Obwohl im obigen Setup Jenkins direkt zum Auslösen einer Bereitstellung verwendet werden kann, stellt das nicht die beste Lösung dar. Da Jenkins eher ein Tool für kontinuierliche Integration ist, ist der Einsatz im Konfigurationsmanagement und für die kontinuierliche Software-Verteilung begrenzt. Alternativ kann für diese Prozesse ein anderes Tool, wie z. B. Ansible oder Chef, gewählt werden. Ansible bietet sich besonders für die Automation von Vorgängen an, die sonst manuell und mit erheblichem Zeitaufwand und ohne eine ausreichende Qualitätskontrolle ausgeführt werden müssten (Red Hat Ansible, o. D.-b). Das folgende Diagramm zeigt einen CI/CD-Setup, bei dem Ansible zur kontinuierlichen Software-Verteilung verwendet wird.

Abbildung 21: Kontinuierliche Integration und Bereitstellung anhand von Git, Jenkins, Ansible, Docker und Kubernetes



Quelle: Kobdani (2020).

Ansible ist eine Software für die zentrale Verwaltung von verteilten Servern. Die Community-Version von Ansible ist lizenzenfrei als Open-Source-Software in der Linux-Verwaltung erhältlich. Neben der Community-Ausgabe gibt es auch andere Ausgaben von Ansible direkt vom Hersteller (z. B. Redhat), die lizenziert sind und beispielsweise Dashboards oder Workflows anbieten. Ansible ist mit Puppet und Chef eines der bekanntesten Softwareprodukte zur Verwaltung von verteilten Servern. Im Gegensatz zu Puppet und Chef hat Ansible jedoch die folgenden Vorteile (Arora, 2020):

- Für Ansible ist keine zentrale Komponente nötig. Ein Computer reicht zum Zugriff auf Server aus, die über SSH (bzw. das Netzwerkprotokoll für verschlüsselte Verbindungen, Secure Shell) gemanagt werden müssen.
- Der Lernprozess ist bei Ansible wesentlich niedriger als bei Chef oder Puppet.
- Für Ansible gibt es viele Skripts (sogenannte Playbooks), die fast alle umsonst (z. B. von GitHub) heruntergeladen werden können.

Folgende Komponenten sind für die Funktion von Ansible notwendig:

- **Workstation/Server:** Im regelmäßigen Einsatz sollte Ansible auf einem Computer oder Server verwendet werden, auf dem Linux installiert ist. Das könnte sowohl die Workstation des Linux-Administrators als auch ein anderer Computer sein, von dem aus die zu verwaltenden Server erreicht werden können.

- **Netzwerk:** Die Server, die von Ansible von der Verwaltungsinstallation verwaltet werden sollen, müssen über ein Netzwerk erreichbar sein. Dabei spielt es keine Rolle, ob dies über das Internet, eine LAN- oder eine VPN-Verbindung geschieht.
- **SSH-Schlüssel:** Die Kommunikation zwischen Ansible und den entfernt gelegenen Hosts findet hauptsächlich über SSH statt. Für den Zugriff von Ansible vom zentralen Host aus auf die entfernten Server ohne Passwort muss eine SSH-Verbindung mit einem Zertifikat existieren (die entsprechende Einrichtung wird unten noch erläutert).

Ansible-Playbooks verwenden in der Regel Yaml und eignen sich besonders gut für die Bereitstellung von komplexen Anwendungen, da sie die Konfigurationsverwaltung erleichtern und ein Framework für die Bereitstellung auf mehreren Computern, das mehrmals wiederholbar ist, ermöglichen. Das folgende Playbook beschreibt beispielsweise die Installation von NGINX:

Code

```
- hosts: local
tasks:
- name: Install Nginx
apt: pkg=nginx state=installed update_cache=true
```

Terraform

Ansible, Puppet und Chef sind Konfigurationsmanagementtools für die Installation und die Verwaltung von Software auf bestehenden Servern. Sie können nicht zum Provisioning eingesetzt werden, wo der Hauptfokus auf der Erstellung, der Änderung und der Versionierung der Infrastruktur liegt. Terraform ist als Provisioning-Software bekannt, d. h. es ist speziell für die Bereitstellung von Servern selbst sowie für die restliche Infrastruktur, z. B. Load Balancer, Datenbanken, gedacht (Terraform, o. D.). Diese Spezialisierung heißt, dass Terraform für bestimmte Aufgaben besonders gut geeignet ist (Brikman, 2016). Immer mehr Unternehmen verwenden heute Cloud-Lösungen zur Implementierung von Arbeitsumgebungen oder sogar für gesamte IT-Strukturen (Rimol, 2019). Infrastructure-as-a-Service ist oft die einfachste und kostengünstigste Lösung als Basis für geplante Projekte. Zudem können aktuelle Anforderungen anhand von Cloud-Lösungen schnell umgesetzt werden. Während die zugrundeliegenden Komponenten (wie Server, Firewalls oder Load Balancer) im Datencenter des Anbieters statisch sind, können sie in der virtuellen Clouddumgebung dynamisch geändert werden. Dadurch hat der:die Kund:in die Möglichkeit, je nach Bedarf mehr oder weniger Ressourcen zu nutzen. Anbieterne bieten diese Flexibilität anhand von APIs, durch die geleasten Infrastrukturumgebungen jederzeit mit der entsprechenden Software skalierbar sind. Diese Anpassungsfähigkeit ist natürlich sehr attraktiv, bedarf jedoch eines hohen Verwaltungsaufwands. Terraform ist die ideale Lösung, um diesen Aufwand langfristig zu minimieren.

2.5 Skalierbare Umgebungen

Skalierbarkeit ist ein Kernziel der modernen Infrastruktur, da eine Organisation, die konsequent skalieren kann, ein großes Wachstumpotenzial hat. Skalierbarkeit bedeutet, dass ein Unternehmen Systeme so konfigurieren kann, dass sie bei hohem Bedarf ausbaufähig sind, und wenn der Bedarf zurückgeht, zurückgefahren werden können. Einige für DevOps typische Aktivitäten bieten sich zur Erreichung optimaler Skalierbarkeit an. Dazu gehören Kommunikationsmöglichkeiten, die Ausrichtung auf Hochleistung, mehr Möglichkeiten für Kreativität und schnellerer Anwendungs-Release. Container-Orchestrierung ist bei Höchstauslastung ein Muss. Durch Container-Orchestrierung werden Provisioning, Verwaltung, Skalierung und Networking von Containern automatisiert. Unternehmen, die hunderttausende von Containern und Hosts bereitstellen und verwalten müssen, kommt die Container-Orchestrierung besonders zugute (Red Hat Ansible, o. D.-c). Die Container-Orchestrierung lässt sich in jeder Umgebung, in der Container eingesetzt werden, verwenden. Mit ihrer Hilfe lassen sich Anwendungen in verschiedenen Umgebungen ohne besondere Anpassung bereitstellen. Mit Microservices in Containern können Dienste, wie Speicherung, Networking und Sicherheit, leichter arrangiert werden (VMware, o. D.).

Folgende Aufgaben können mithilfe der Container-Orchestrierung automatisiert und verwaltet werden:

- Provisioning und Bereitstellung,
- Konfigurierung und Planung,
- Ressourcenzuordnung,
- Containerverfügbarkeit,
- Skalierung und Entfernung von Containern zur ausgeglichenen Auslastung in der Infrastruktur,
- Lastenausgleich und -routing,
- Überwachung des Containerstatus,
- Konfigurierung von Anwendungen nach den Containern, in denen sie ausgeführt werden sollen, sowie
- Absicherung der Containerinteraktion.

Tools zur Container-Orchestrierung bieten ein Framework für die groß angelegte Verwaltung von Containern und Microservicearchitekturen. Es gibt viele Container-Orchestrierungs-Tools, die zum Container-Lifecycle Management verwendet werden können. Zu den am häufigsten eingesetzten gehören Kubernetes, Docker Swarm und Apache Mesos (Apache Mesos, o. D.).

Kubernetes

Kubernetes ist ein ursprünglich von Google entworfenes und entwickeltes Open-Source-Tool für die Orchestrierung für Containern. Mit einer Kubernetes-Orchestrierung können Anwendungsdienste über mehrere Container hinweg entwickelt werden, Cluster-übergreifend Containerplanung und -skalierung vorgenommen und ihre Integrität auf Dauer überwacht werden (Kubernetes, o. D.). Viele der mit der Bereitstellung und Skalierung von Containeranwendungen verbundenen, manuellen Prozesse fallen mit Kubernetes weg.

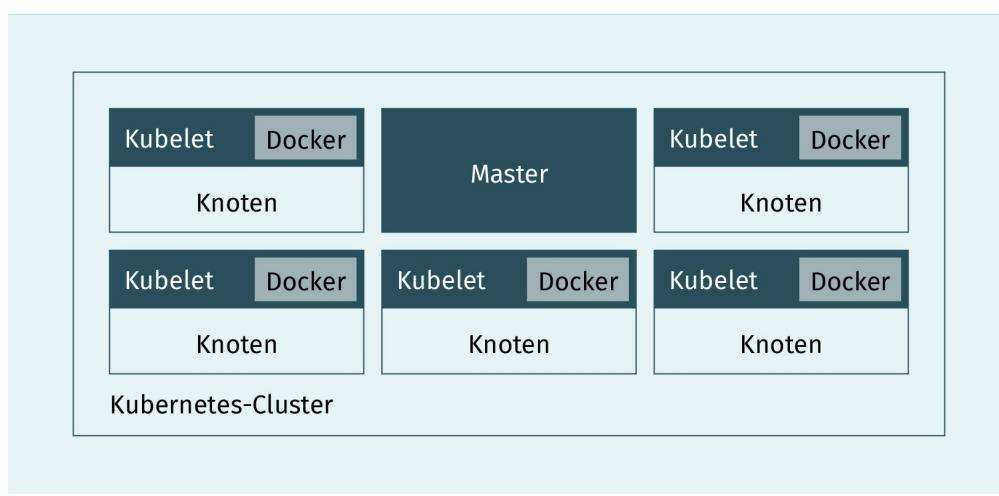
Container können auf Cluster-Host-Gruppen aus physischen oder virtuellen Maschinen ausgeführt werden, da Kubernetes eine gute Plattform für die einfache und effiziente Verwaltung dieser Cluster bietet.

Cloud Native

Ein Ansatz, bei dem gewährleistet wird, dass sich Design und Entwicklung einer Anwendung für Cloud Computing eignen.

Im Allgemeinen kann man mit Kubernetes in einer Produktionsumgebung eine komplett Container-basierte und zuverlässige Infrastruktur implementieren. Mit Kubernetes sind Cluster mit Hosts in öffentlichen, privaten oder Hybrid **Clouds** möglich. Kubernetes ist ein gutes Framework zum Hosting von cloudbasierten Anwendungen, für die schnelles Skalieren notwendig ist. Workloads sind mit Kubernetes auch portierbar und Lastenausgleich ist möglich, da Anwendungen ohne Neuentwicklung übertragen werden können. Das folgende Diagramm zeigt ein Kubernetes-Cluster:

Abbildung 22: Kubernetes-Cluster



Quelle: Kobdani (2020).

Ein Kubernetes-Cluster besteht aus zwei Ressourcenarten:

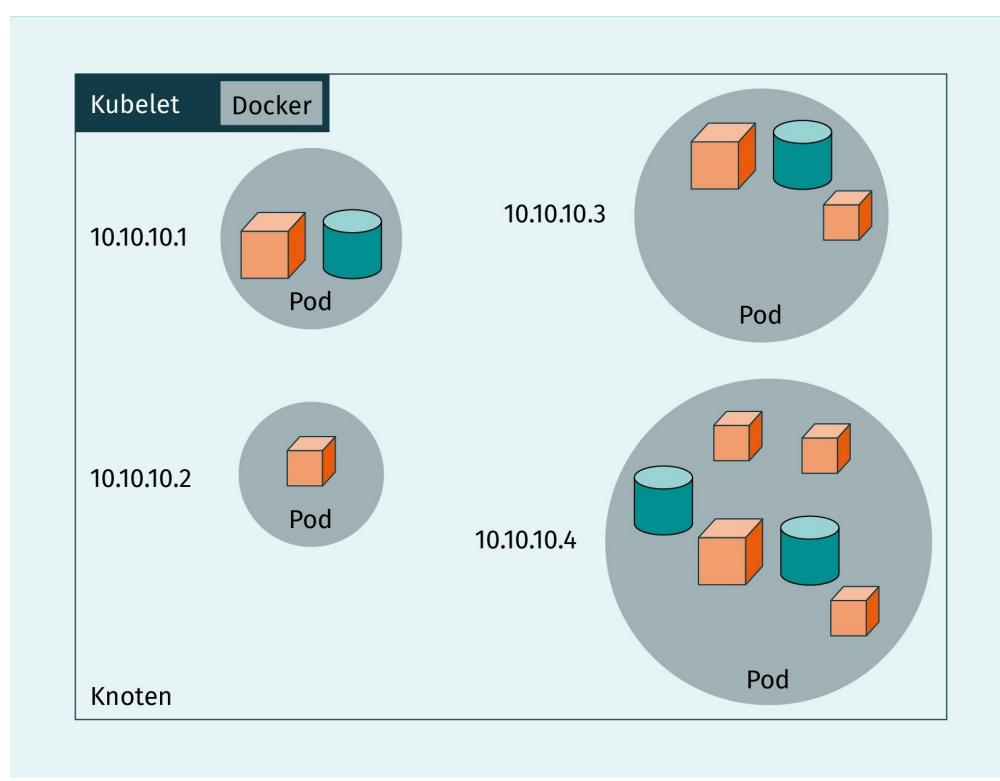
- Master, d. h. einem Clusterkoordinator;
- Knoten, d. h. den Workern, die die Anwendungen ausführen.

Der Master ist für die Verwaltung des Clusters verantwortlich. Über den Master laufen alle Aktivitäten im Cluster, z. B. Anwendungsplanung, Verwaltung des gewünschten Anwendungsstatus, Anwendungsskalierung und Implementierung neuer Updates. Ein Knoten ist ein virtueller oder ein physischer Computer, der als Arbeitsmaschine in einem Kubernetes-Cluster eingesetzt wird. Jeder Knoten hat einen Agenten zur Verwaltung des Knotens und zur Kommunikation mit dem Kubernetes-Master, das sogenannte Kubelet. Der Knoten sollte auch Tools zur Handhabung des Containerbetriebs, wie z. B. Docker, haben. Ein für die Produktionslast verantwortlicher Kubernetes-Cluster sollte mindestens drei Knoten haben. Wenn Anwendungen auf Kubernetes bereitgestellt werden, muss dem Master der Befehl zum Starten der Anwendung gegeben werden. Der Master plant die Ausführung der Container auf den Clusterknoten. Die Knoten kommunizieren mit dem Master über die von diesem zur Verfügung gestellte Kubernetes-API. Benutzer:innen können über die Kubernetes-API auch direkt mit dem Cluster arbeiten. Pods sind die kleinsten Recheneinheiten, die

in Kubernetes erstellt und verwaltet werden können. Ein Pod (abgeleitet vom Englischen „pod“ für Gruppe) ist eine Gruppe aus einem oder mehreren Containern mit gemeinsamen Speicher- und Netzwerkressourcen; Pod ist auch eine Spezifikation für die Ausführung von Containern (Kubernetes, o. D.). Die Inhalte eines Pods befinden sich oft an einem gemeinsamen Speicherort und folgen demselben Zeitplan. Ein Pod ist ein Modell für einen anwendungsspezifischen Logik-Host, der einen oder mehrere, relative eng verbundene Anwendungscontainer enthält. In Umgebungen außerhalb der Cloud gleichen Programme auf derselben physischen oder virtuellen Maschine den Cloud-Anwendungen auf demselben Logik-Host.

Ein Pod wird immer auf einem Knoten ausgeführt. Der Knoten kann mehrere Pods haben. Der Kubernetes-Master kann die Zeitplanung für Pods über alle Knoten in einem Cluster automatisch verwalten. In die automatische Zeitplanung des Masters wird einbezogen, welche Ressourcen für eine Knoten notwendig sind. Im folgenden Diagramm wird dargestellt, wie Pods auf Knoten ausgeführt werden.

Abbildung 23: Kubernetes-Pods



Quelle: Kobdani (2020).

Docker Swarm

Das Konzept von Docker Swarm ist ähnlich wie Kubernetes - beide werden in einem Cluster zur Bereitstellung und Verwaltung von Containern verwendet. Kubernetes und Docker Swarm sind dafür konzipiert, Knoten-Cluster in einer großen Größenordnung in Produktionsumgebungen zu koordinieren. Swarm ist eine von Docker-Entwicklern erstellte Soft-

ware, bei der Docker-Hosts in einem Cluster zusammengeführt werden und zentrales Cluster-Management und Container-Orchestrierung ermöglichen. Bis zur Version 1.11 von Docker musste Swarm als separates Tool implementiert werden, während die neueren Versionen der Containerplattform den Swarm im nativen Modus unterstützen. Der Cluster-Manager ist also nach Installation der Docker-Engine für jeden Docker-Benutzer:in zugänglich (Docker, o. D.-d). Docker Swarm beruht auf einer Master-Slave-Architektur. Jeder Docker-Cluster, der sogenannte Swarm, besteht aus mindestens einem Manager-knoten und beliebig vielen Worker-Knoten. Während der Swarm-Manager für die Verwal-tung der Cluster und die Delegierung von Aufgaben verantwortlich ist, führen die Swarm-Worker die Aufgabeneinheiten aus.

Containeranwendungen werden als Dienst auf so vielen Docker-Konten wie nötig verteilt. In der Docker-Swarm-Terminologie ist ein „Dienst“ eine abstrakte Struktur, die zur Defini-tion von Aufgaben, die im Cluster ausgeführt werden müssen, verwendet wird. Jeder Dienst besteht aus einer Reihe von einzelnen Aufgaben, die allein in einem separaten Container auf den Knoten im Cluster abgearbeitet werden. Bei der Erstellung eines Dienstes werden das zugrundeliegende Containerimage und die im Container auszuführenden Befehle festgelegt. Docker Swarm unterstützt zwei Modi, in denen Swarm definiert ist: replizierte und globale Dienste.

Replizierte Dienste

Bei einem replizierten Service handelt es sich um eine Aufgabe, die in einer benutzerdefinierten Anzahl von Replikaten ausgeführt wird. Jedes Replikat ist eine Instanz des im Service definierten Docker-Containers. Replizierte Services lassen sich skalieren, indem Nutzer:innen weitere Replikate erzeugen. Ein Webserver, wie NGINX, lässt sich beispielsweise je nach Bedarf mit einer einzigen Befehlszeile auf 2, 4 oder 100 Instanzen skalieren.

Globale Dienste

Wird ein Service im globalen Modus ausgeführt, startet jeder verfügbare Knoten im Cluster eine Aufgabe für den entsprechenden Dienst. Wird dem Cluster ein neuer Knoten hinzuge-fügt, teilt der Swarm-Manager diesem unverzüglich eine Aufgabe für den globalen Dienst zu. Globale Services eignen sich beispielsweise für Monitoring- oder Protokollierungsan-wendungen. Ein zentrales Anwendungsfeld von Docker Swarm ist die Lastenverteilung. Im Swarm-Modus verfügt Docker über integrierte Load-Balancing-Funktionen. Wird ein NGINX-Webserver mit 4 Instanzen ausgeführt, verteilt Docker eingehende Anfragen intelli-gent auf die zur Verfügung stehenden Webserver-Instanzen.

Kubernetes und Docker Swarm im Vergleich

Kubernetes unterstützt höheren Bedarf mit mehr Komplexität, während Docker Swarm eine einfache Lösung, die auch für Anfänger leicht verständlich ist, bietet. Docker Swarm ist bei Entwicklern und Entwicklerinnen, die eine schnelle Bereitstellung und Einfachheit schätzen, sehr beliebt. Gleichzeitig wird Kubernetes in Produktionsumgebungen vieler namhafter Internet-Firmen, die geläufige Dienste anbieten, verwendet (Mangat, 2019). Kubernetes und Docker Swarm bieten viele der gleichen Dienste an; je nach speziellem

Bedarf muss allerdings auch ein etwas anderer Ansatz gewählt werden. Durch den Vergleich der beiden Tools lässt sich das richtige für eine bestimmte Container-Orchestrierung auswählen.



ZUSAMMENFASSUNG

In dieser Lektion wurden DevOps, die Schwierigkeiten von Entwicklungs- und Betriebsteams in klassischen Umgebungen und die Abhilfe, die DevOps in solchen Situationen bieten kann, behandelt. Auch wichtige Aspekte der DevOps-Kultur, wie die DevOps-Pipeline, das Konfigurationsmanagement, die Releaseverwaltung, die kontinuierliche Integration und die kontinuierliche Lieferung, „Infrastructure as Code“, die Testautomatisierung und das kontinuierliche Monitoring wurden erläutert. Die Auswirkungen von DevOps auf die Team- und Entwicklungsstrukturen wurden anhand von Teamgröße, Teamführung, Microservice-Architektur und den Prinzipien der Twelve-Factor-App dargelegt.

Diese Lektion lieferte zudem eine kurze Einführung in den Aufbau einer DevOps-Infrastruktur und deren Aufskalierung und bot einen Überblick über CI/CD-Pipelines mit Git, Jenkins, Ansible, Docker und Kubernetes.

LEKTION 3

SOFTWAREENTWICKLUNG

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- verschiedene Testverfahren im Lebenszyklus der Softwareentwicklung zu nennen.
- verschiedene Testansätze während der Entwicklung zu nennen, wie die testgetriebene und die verhaltensgetriebene Softwareentwicklung.
- kontinuierliche Integration (CI), kontinuierliche Lieferung (CD), kontinuierliches Testen (Cte) und kontinuierliches Training (CT) zu erklären.
- die Verwendung eines CI/CD-Ansatz zur Verbesserung von Machine-Learning-Systemen zu erklären.
- Änderungen in einem Softwareentwicklungsprojekt mithilfe von Tools nachzuverfolgen.
- integrierte Entwicklungstools (IDEs) zu erkennen und einige Beispiele zu nennen.

3. SOFTWAREENTWICKLUNG

Einführung

Nach Abschluss der Entwicklung einer Machine-Learning-Lösung (ML) wird die Software nicht sofort den Kund:innen in der Produktionsumgebung zur Verfügung gestellt. Davor erfolgt noch ein weiterer Schritt, und zwar das Testen. Testen gilt als eine funktionsübergreifende und kontinuierliche Aktivität des ganzen Teams. Im Idealfall arbeitet das Testteam mit den Entwickler:innen und Benutzer:innen zusammen, um automatisiertes Testen bereits in der Anfangsphase des Projekts zu entwickeln und zu verbessern. Diese automatisierten Tests werden vom Testteam vor der Entwicklung der Module durch das Entwicklungsteam erstellt. Wenn die entwickelten Module die relevanten Tests bestehen, zeigt das, dass die notwendige Funktionalität in der Software vorhanden ist. Der Vorgang, bei dem die Projektrisiken ermittelt und nach ihrer Priorität geordnet werden sowie die Risikominderungsmaßnahmen festgelegt wurden, ist die Teststrategie. Eine gute Teststrategie führt zu einer funktionierenden Software (d. h. weniger Programmfehler und geringere Supportkosten), und legt ein Framework für Entwicklungspraktiken fest. Am Anfang dieser Lektion werden verschiedene Testarten aus der Sicht der verschiedenen Stakeholder:innen eines Projekts besprochen. Im Anschluss befassen wir uns damit, wie das Testen und Entwickeln gleichzeitig stattfinden kann. Ein Szenario für die Entwicklung einer Machine-Learning-Lösung, also einem nicht-deterministischen System, ist äußerst komplex, weshalb die konventionellen Testverfahren zuvor angepasst werden müssen. Zudem wird die Automatisierung der Entwicklungspipeline behandelt. Darüber hinaus besprechen wir die automatisierte Entwicklung, bei der es noch bedeutsamer ist, Änderungen nachzuverfolgen.

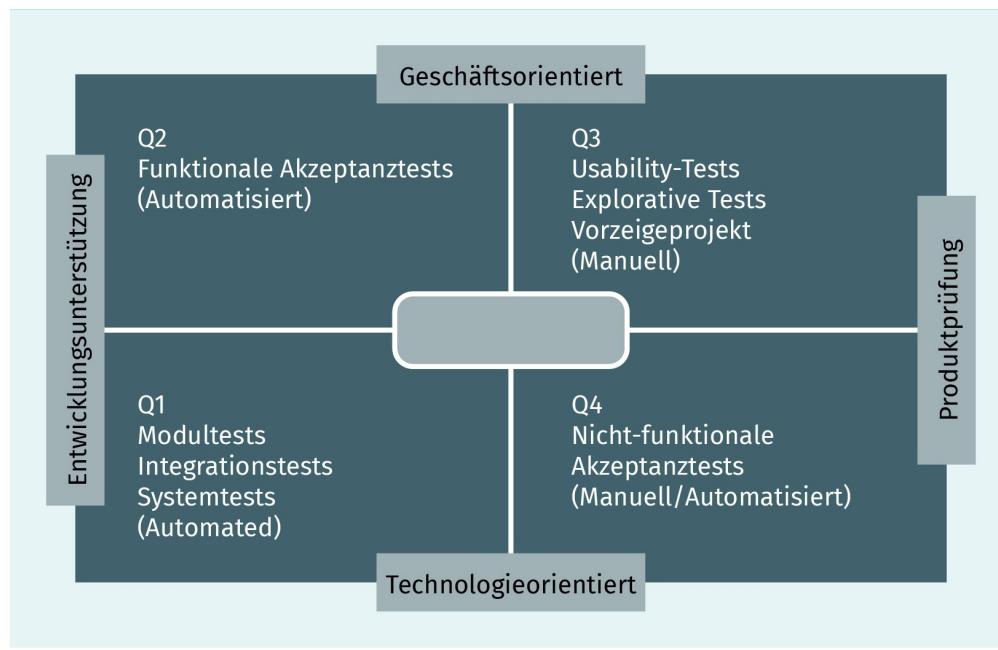
3.1 Testparadigmen und Überwachung

Brian Marick entwickelte verschiedene Testszenarien, um hochwertige Software-Lösungen zu erzielen. Sie werden im folgenden Testquadranten dargestellt. Ein Projektteam sollte diese implementieren und vor dem Release ausführen (Humble & Farley, 2015). Diese Tests lassen sich in zwei Dimensionen unterteilen:

1. Tests zur Unterstützung der Entwicklung, durch welche die Software mit der nötigen Sicherheit erstellt werden kann, und Tests zur Beurteilung des Produkts, durch die Unzulänglichkeiten und Mängel am Programm entdeckt werden können.
2. Tests für die Geschäftsbereiche, die aus Sicht einer Geschäftsperson entwickelt wurden, und Tests für die Entwicklung, die aus der Perspektive eines: einer Entwickler:in erstellt wurden.

Diese vier Szenarien werden nun im Detail behandelt.

Abbildung 24: Testquadrant nach Marick



Quelle: Alvarid (2020), in Anlehnung an Humble (2015).

Quadrant 1 (Q1)

Zu den technologieorientierten Tests für das Entwicklungsteam gehören Modultests, Integrationstests und Systemtests. Sie werden normalerweise vom Entwicklungsteam geschrieben. In einem Modultest wird ein Softwaremodul in einer isolierten Umgebung getestet. Die Interaktion mit den anderen Softwarekomponenten (Modulen) wird simuliert, denn Input ist notwendig. Durch die Isolation kann das Modultesten schnell abgewickelt werden. Allerdings kann durch den Modultest nicht die tatsächliche Funktionalität im Zusammenspiel mit anderen Softwarekomponenten getestet werden. Daher muss eine andere Testreihe, die Integrationstests, angesetzt werden. Bei einem Integrationstest wird die Funktionalität in verschiedenen Modulen des Softwareprodukts getestet. Dabei werden nicht nur die Modulfunktionalität, sondern auch die Interaktionen und die Schnittstellen der Module getestet. Durch Integrationstesten wird gewährleistet, dass jedes unabhängige Softwaremodul mit den dafür notwendigen Diensten problemlos funktioniert. Ein Integrationstest des zu testenden Systems könnte gegen die tatsächlichen externen Systeme, die notwendig sind, oder in einem **Testharnisch**, d. h. einer Testumgebung, die Teil des Softwareprodukts ist, ausgeführt werden. Beim Systemtesten wird das Softwareprodukt als Ganzes geprüft, um die Kompatibilität mit den Kund:innenanforderungen, die in der Software-Anforderung (kurz SRS für Software Requirements Specification) festgelegt ist, zu beurteilen. Bei Integrationstests werden ausgewählte Module und deren Interaktion mithilfe von Funktionstests geprüft. Beim Systemtesten wird das System als Ganzes getestet. Es erfolgt im Anschluss an die Integrationstests. Vor der Bereitstellung einer Software-Lösung in der Produktionsumgebung ist zudem ein Bereitstellungstest notwendig. Dabei wird beispielsweise geprüft, ob die Software richtig installiert und konfiguriert wurde und ob sie mit den notwendigen Diensten kommunizieren kann.

Testharnisch

Darunter wird in der Qualitätssicherung von Software eine Sammlung von Software- und Testdaten verstanden, die zum automatisierten Testen eines Programms unter verschiedenen Umgebungsbedingungen und zur Überwachung des Verhaltens und des Outputs verwendet wird.

Quadrant 2 (Q2)

Funktionaler Akzeptanztest

Anforderungen, in denen festgelegt ist, was von einem Softwaresystem verlangt wird bzw. was nicht gewünscht wird, werden als funktionale Anforderungen bezeichnet. Nicht-funktionale Anforderungen legen fest, wie das System funktionieren soll.

Tests, die das Entwicklungsteam in geschäftlichen Aspekten unterstützt, werden als **funktionale Akzeptanztests** bezeichnet. Ein Akzeptanztest gewährleistet, dass vorgegebene Kriterien, wie Funktionalität, Kapazität, Benutzerfreundlichkeit und Sicherheit, erfüllt werden. Ein Akzeptanztest für eine bestimmte Funktion wird als funktionaler Akzeptanztest bezeichnet. Wenn eine Software einen derartigen Test besteht, können das Entwicklungsteam und die Benutzer:innen davon ausgehen, dass die Funktionalität des entwickelten Moduls richtig implementiert wurde. Zu dieser Kategorie gehört auch das End-to-End-Testen (E2E-Testen), bei dem Entwickler:innen prüfen, ob die gesamte Software von Anfang bis Ende wie erwartet funktioniert (Q2). Bei der Testkonzipierung werden die Systemabhängigkeiten definiert. Der Test gewährleistet, dass alle Teile der Softwarelösung gemäß den Vorgaben zusammenwirken. Das Hauptziel des E2E-Testen besteht darin, die Software durch die Simulation tatsächlicher Benutzerszenarien aus der Sicht der Benutzer:innen zu testen (Katalon, 2020). Beim Systemtesten wird nur das Softwaresystem selbst getestet. Beim E2E-Testen hingegen wird das System in Verbindung mit den externen Systemen getestet.

Quadrant 3 (Q3)

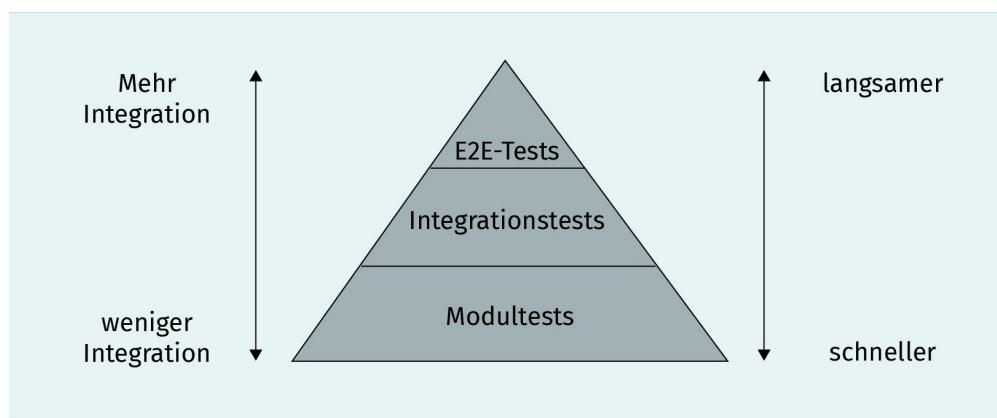
Geschäftsorientierte Tests zur Beurteilung des Projekts sind meist manuelle Tests, durch die geprüft wird, ob das Produkt den von Kund:innen erwarteten Wert liefern kann. Dabei werden nicht nur die Spezifikationen, sondern auch die Definition der Spezifikationen geprüft. In diese Testkategorie fallen Showcases, explorative Tests und Tests zur Überprüfung der Benutzerfreundlichkeit. Am Ende einer Iteration stellt das Entwicklungsteam den Benutzer:innen die neue Funktionalität im Rahmen eines Musterbeispiels vor. Diese Funktionalitätsdemo während der Entwicklungsphase beugt Missverständnissen oder Spezifikationsproblemen vor. Ein weiteres Testverfahren, das explorative Testen, wurde von J. Bach (2003) als manuelles Testen definiert, bei dem „**Testingenieur:innen** aktiv das Testdesign während der Testausführung kontrollieren und die gewonnenen Informationen zur Entwicklung neuer, verbesserter Tests verwendet“ (Bach, 2003, S. 2, vom Autor übersetzt). Exploratives Testen führt zu einer Reihe von neuen automatisierten Testverfahren. Abschließend werden Tests zur Überprüfung der Nutzerfreundlichkeit durchgeführt, um nachzuholen, wie Benutzer:innen anhand der entwickelten Software die definierten Ziele erreichen können.

Quadrant 4 (Q4)

Technologieorientierte Tests zur Beurteilung eines Projekts verifizieren die nicht-funktionellen Kriterien des Softwaresystems, z. B. die Kapazität, Verfügbarkeit und Sicherheit. Dazu wurde das nicht-funktionelle Akzeptanztesten entwickelt. Diese Testart könnte voll automatisiert werden, sie wird allerdings seltener als das funktionale Akzeptanztesten und erst am Ende der Entwicklungspipeline ausgeführt. Nachdem wir die verschiedenen Softwaretestverfahren angesprochen haben, können wir nun auf die Frage eingehen, wie automatisierte Testreihen angelegt werden. Eine effektive automatisierte Teststrategie kann auf drei Ebenen aufgeteilt werden. Das ist die Struktur der sogenannten Testautomationspyramide, wie nachfolgend dargestellt (Cohn, 2009). Die Grundlage der Pyramide bilden in diesem Modell die Modultests. Die nächste Ebene bildet das Integrationstesten, und

an der Spitze der Pyramide befindet sich das End-to-End-Testen. Die Testpyramide ist die ursprüngliche Darstellung dieses Ansatzes, der um weitere automatisierte Testverfahren (hauptsächlich aus Quadrant 1 und Quadrant 2 des Testquadranten nach Marick) erweitert wird.

Abbildung 25: Testautomationspyramide



Quelle: Alvarid (2020), in Anlehnung an Humble (2015).

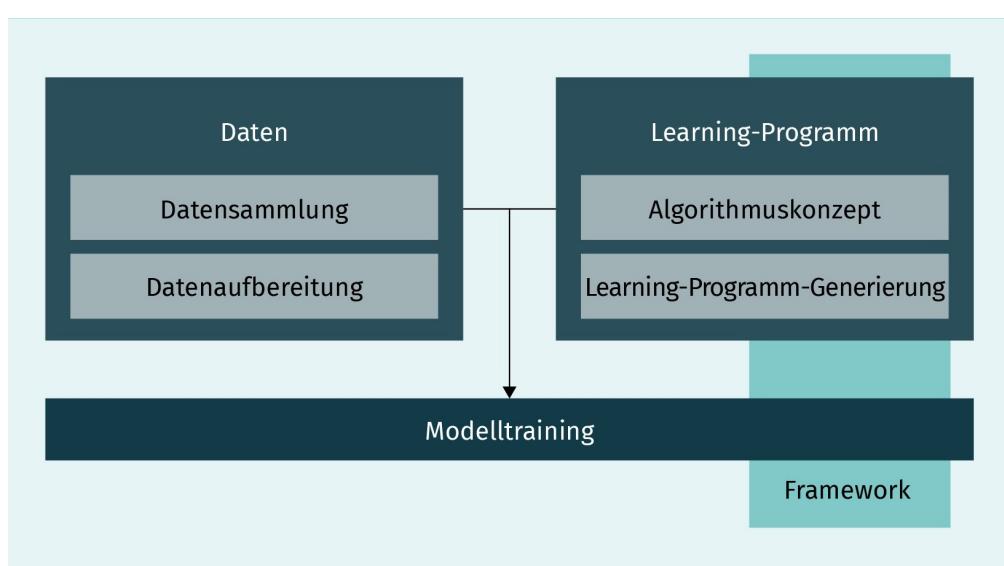
Testen von Machine-Learning-Modellen

Im Gegensatz zu herkömmlichen Softwareprodukten sind Machine-Learning-Modelle (ML) **nicht-deterministisch**. Es gibt jedoch mehrere Möglichkeiten für den Übergang in den nächsten Zustand mit demselben Input. Denn bei einer derartigen Software-Lösung hängt die Reaktion des Systems davon ab, was das System in den vorherigen Transaktionen gelernt hat. Daher sollte der:die Tester:in bei ML-Lösungen sowohl die Testdaten, den Code, das Lernprogramm als auch das zur Entwicklung der ML-Lösung verwendete Framework (z. B. TensorFlow) testen. Zudem ist die Entwicklung eines Testorakels zeit- und arbeitsaufwändig, denn dazu ist fachspezifisches Wissen notwendig (Zhang et al., 2020). Ein Orakel ist ein beim Softwaretesten und -entwickeln verwendeteter Mechanismus, um den Erfolg des Tests zu messen (Kaner, 2004). Dabei wird das Output des zu testenden Systems, das als Input für ein bestimmtes Testszenario vorbereitet wurde, mit dem Ergebnis verglichen, das das Produkt (nach Einschätzung des Orakels) liefern sollte.

Nichtdeterminismus
Ein Begriff aus der Computerwissenschaft, bei dem Algorithmen oder Maschinen nicht nur genau eine Berechnung auf einen bestimmten Input hin (deterministisch) durchführen können. Es gibt dagegen mehrere Möglichkeiten für den Übergang zum nächsten Zustand für den gleichen Input.

Die Komponenten, die zur Erstellung eines Machine-Learning-Modells notwendig sind, werden in folgender Abbildung dargestellt (Zhang et al., 2020). Beim Testen einer ML-Lösung sollten Tester:innen alle genannten Komponenten und die entsprechenden Interaktionen testen.

Abbildung 26: Komponenten für ein ML-Modell



Quelle: Alvarid (2020), in Anlehnung an Zhang et al. (2020).

Testen von Daten

Beim Testen von Datenkomponenten eines ML-Modells sollten Testingenieur:innen beispielsweise prüfen, ob ausreichend Daten zum Training vorhanden sind, ob die Daten repräsentativ für zukünftige Daten sind, ob die Daten viel Noise (z.B. verzerrte Daten) enthalten, und ob zwischen den Trainings- und Testdaten eine Verzerrung (zu Deutsch auch Bias genannt) besteht (Zhang et al., 2020). Obwohl es die Aufgabe des:der ML-Entwicklenden ist, die richtigen Testfälle zu konzipieren, gibt es allgemeine Richtlinien (Heck, 2020):

- **Datentyp und -schema:** ML-Ingenieur:innen überprüfen Format, Art und Schema der Daten stets in Hinblick auf Datenqualität und -integrität. Anhand dieses Tests wird sichergestellt, dass die Daten keine schlecht formatierten Inputs enthalten (Kim et al., 2018).
- **Implizite Constraints:** ML-Ingenieur:innen untersuchen häufig bestimmte Einschränkungen, sogenannte Constraints. Dabei handelt es sich nicht um einzelne Datenpunkte, sondern um Datenuntergruppen, die sich auf andere Untergruppen beziehen (Kim et al., 2018). Zum Beispiel sollte die Zahl der heruntergeladenen Softwarelizenzen aus einer bestimmten Region der Zahl der gekauften Lizenzen entsprechen.
- **Testplattformen:** Große Firmen bieten ML-Test-Frameworks an, z. B. TFX von Google (Breck et al., 2019) und Apple Overton (Ré et al., 2019). Google bietet beispielsweise ein Datenvierifizierungssystem an, anhand dessen Datenschemata zur Prüfung von Dateneigenschaften definiert und künstliche Daten zum Testen des Modells generiert werden können (Breck et al., 2019).

Testen des lernenden Programms

Ein lernendes Programm, auch Learning-Programm und ML-Modell genannt, besteht aus Algorithmen und dem Code zur Implementierung und Konfigurierung. Ein trainiertes ML-Modell sollte nach Metriken wie Richtigkeit, Fairness und Interpretierbarkeit getestet werden (Zhang et al., 2020). Zur Richtigkeit gehören Accuracy (Genauigkeit), Precision (Präzision) und Recall (Abruf). Der Unterschied zwischen Precision und Recall kann am besten am Beispiel eines ML-Algorithmus erklärt werden.

Der Algorithmus teilt 100 Tumore in die Klassen bösartig (die positive Klasse) oder gutartig (die negative Klasse) ein. Daraus ergeben sich richtig positiv (TP), falsch positiv (FP), falsch negativ (FN) und richtig positiv (TN) (Google, 2020a). Precision ist das Verhältnis aus tatsächlich richtigen positiven Ergebnissen ($TP/(TP+FP)$). Ein Recall ist der Anteil der tatsächlich positiven Fälle, die vom Algorithmus richtig identifiziert wurden ($TP/(TP+FN)$). Ein fairen ML-Modell liefert faire Vorhersagen über verschiedene demographische Gruppen hinweg (Dwork et al., 2012). IBM entwickelte ein Tool für ML-Fairness, das AI Fairness 360 (IBM, o. D.-a). Es wird von ML-Entwickler:innen verwendet, um Diskriminierung und Zerstörung in ML-Modells im gesamten AI-Anwendungslebenszyklus zu prüfen, zu protokollieren und auszugleichen. Ein ML-Modell kann als interpretierbares Modell verstanden werden, wenn ein:e Beobachter:in bzw. Benutzer:in den Grund der vom Modell getroffenen Entscheidungen verstehen kann. Es gibt Tools zur Interpretierbarkeitsprüfung, z. B. das Modul Machine Learning Interpretability von H2O (H2O, o. D.).

ML-Modell-Überwachung

Ein ML-System sollte in Hinblick auf unerwartetes Verhalten aufgrund von Änderungen an den Inputdaten beobachtet werden. Überwachung oder Monitoring ist für Modelle unerlässlich, die automatisch und in Echtzeit neue Daten während des Trainings aufnehmen. Daher müssen die Vorhersagen solcher Modelle ebenfalls in Echtzeit stattfinden. Es gibt vier Klassen von ML-Monitoringsystemen, die folgendermaßen beschrieben werden können (Gade, 2019):

1. Funktionsüberwachung zur Sicherstellung der langfristigen Stabilität eines Modells und der Datenvarianten sowie zur ständigen statistischen Überwachung;
2. Modellbetriebsüberwachung zur Ermittlung von veralteten Daten, Regression der Latenzzeit, Durchsatz, RAM-Verbrauch usw.;
3. Modellleistungsüberwachung zur Feststellung von Regressionen bei der Vorhersagequalität;
4. Bias-Überwachung des Modells zum Fund von unbekannten Bias.

Das Amazon SageMaker Model Monitor ist ein Beispiel für ein ML-Monitoring-Tool, anhand dessen Entwickler:innen das Phänomen des **Konzeptdrifts**, bei dem sich Daten ständig ändern, erkennen und beheben (Amazon, o. D.-b). SageMaker entdeckt in bereitgestellten ML-Modellen Konzeptdrift und liefert detaillierte Benachrichtigungen, die bei der Erkennung der Problemursache helfen.

Konzeptdrift

Wenn sich die Vorhersagedaten von den Trainingsdaten unterscheiden, stimmen die Muster aus der Vorhersage nicht mehr. Dann spricht man von Konzeptdrift.

3.2 Entwicklungs- und Testansätze

In der Softwareentwicklung wird eine Entwicklungsmethode, die auch als Lebenszyklus bezeichnet wird, auf die Softwareentwicklungsprojekte angewandt, die in separate Phasen oder Stufen unterteilt werden. Jede Phase umfasst bestimmte Aktivitäten mit dem Ziel, die Planung und Verwaltung des Projekts effizienter zu gestalten. Das Wasserfallmodell ist eine der klassischen Methoden, während **agile Modelle**, wie Kanban und Scrum, modernere Ansätze darstellen (Lumen, 2020). In diesem Abschnitt wird kurz auf die drei Softwareentwicklungstechniken eingegangen, durch die agile Entwicklungsmethoden unterstützt werden, indem Testpraktiken und automatisiertes Testen optimiert werden:

- Testgetriebene Entwicklung (TDD)
- Verhaltensgetriebene Entwicklung (BDD)
- Akzeptanztest-getriebene Entwicklung (ATDD)

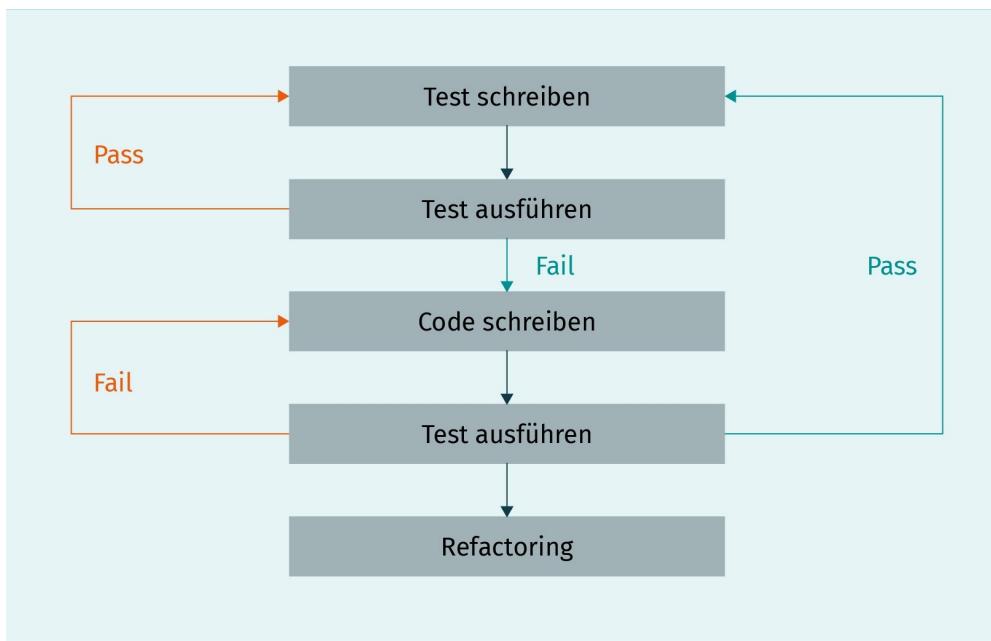
Testgetriebene Entwicklung (TDD)

Bei der testgetriebenen Entwicklung werden Tests aus der Sicht des:der Entwickler:in ausgeführt. Dafür entwerfen und schreiben Qualitätssicherungsingenieur:innen zunächst Testfälle für alle Funktionen der Software. Dabei soll der einfachen Frage nachgegangen werden, ob der Code gültig ist. Der Hauptzweck dieser Technik besteht darin, den Code nur dann zu ändern oder neu zu schreiben, wenn der Test fehlschlägt. Dadurch wird die Duplikation von Testskripts vermieden. Diese Technik wird häufig in der agilen Entwicklung verwendet. Bei einem TDD-Ansatz werden funktionale Codeteile erst nach der Erstellung der automatisierten Testskripts geschrieben. Der TDD-Lebenszyklus lässt sich in folgende Schritte unterteilen (Beck, 2014):

1. Test schreiben; dieser sollte bestimmte Funktionen definieren (oder verbessern). Daher sollten Testentwickler:innen die Produktanforderungen kennen. Dies unterscheidet den klassischen Entwicklungsansatz von testgetriebener Entwicklung: denn erst wird der Test und daraufhin der Code geschrieben.
2. Test ausführen und sicherstellen, dass dieser fehlschlägt, denn es gibt schließlich noch keine Funktionalität.
3. Code schreiben und zwar für die Funktionalität, die den bereits geschriebenen Test bestehen soll. Der Code sollte so einfach geschrieben sein, dass der Test in jedem Fall funktioniert.
4. Test am neu geschriebenen Code ausführen. Er sollte den Test bestehen.
5. **Refaktorierung** des Codes, wobei doppelt geschriebener Code entfernt und der in der testgetriebenen Entwicklung geschriebene Code bereinigt wird. Auch die Namenskonventionen sollten dabei geprüft und gegebenenfalls korrigiert werden.

Im folgenden Flussdiagramm werden die obigen Schritte dargestellt:

Abbildung 27: Lebenszyklus der testgetriebenen Entwicklung

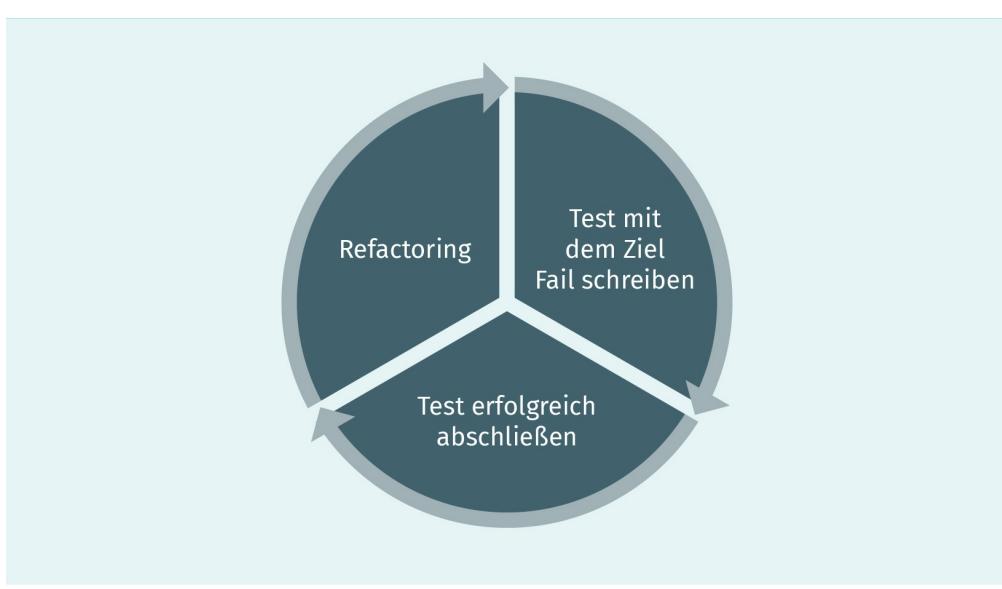


Quelle: Alvarid (2020).

Zusammenfassend lässt sich sagen, dass bei der testgetriebenen Entwicklung wie folgt vorgegangen wird:

1. einen fehlschlagenden Test schreiben,
2. den Test so verändern, dass dieser bestanden wird,
3. Refaktorierung und
4. Wiederholung dieser Schleife (Freeman & Pryce, 2012).

Abbildung 28: TDD-Zyklus



Quelle: Alvarid (2020).

Verhaltensgetriebene Entwicklung (BDD)

Beim Einsatz von testgetriebener Entwicklung in Projekten in verschiedenen Umgebungen kann es zu Verwirrung und Missverständnissen kommen. Programmierer:innen wissen möglicherweise nicht, wo sie beginnen sollen, was getestet werden muss und was nicht, wie viel auf einmal getestet werden muss, wie sie den Test nennen sollen, und warum ein Test nicht bestanden wird (North, 2020). Ein Ansatz, um dem gegenzusteuern, ist die verhaltensgetriebene Entwicklung (BDD), eine agile Softwareentwicklungspraktik nach Dan North (2006). Dabei wird angestrebt, dass alle Beteiligten verstehen, wie eine Anwendung funktionieren sollte, indem neue Funktionen an konkreten Beispielen erarbeitet werden. Diese typischen Beispiele werden in natürlicher Sprache in der Struktur „Gegeben/Wenn/Dann“ geschrieben.

- **Gegeben:** Vorläufiger Kontext, bekannte Tatsachen.
- **Wenn:** Ein Ereignis passiert.
- **Dann:** Ein bestimmtes Ergebnis folgt.

Eine Implementierung des „Gegeben/Wenn/Dann“-Ansatzes könnte folgendermaßen aussehen.

- **Gegeben:** Der:Die Benutzer:in hat gültige Anmeldeinformationen eingegeben.
- **Wenn:** Benutzer:in klickt auf die Anmeldeschaltfläche.
- **Dann:** Die erfolgreiche Validierung wird durch eine Benachrichtigung angezeigt.

Durch die Verwendung von Klartext (hier in Deutsch) können alle Stakeholder:innen (Entwickler:innen und Kundschaft) das Funktionsverhalten nachvollziehen. Gherkin ist eine fachspezifische Sprache für Beispiel in der „Gegeben/Wenn/Dann“-Struktur und in Klar-

text-Dateien, die als Feature-Dateien bezeichnet werden. Feature-Dateien beschreiben, wie System und Benutzer:innen miteinander interagieren. Ein einfaches Beispiel einer Feature-Datei in Gherkin könnte folgendermaßen aussehen:

Code

```
Feature: Rechner
Szenario: "+" sollte zur laufenden Summe addiert werden
Gegeben ist die laufende Summe "5"
Wenn ich "7" eingebe
Dann sollte die laufende Summe "12" sein
```

Das Scenario im obigen Beispiel beschreibt einen Schritt in einer „Gegeben/Wenn/Dann“-Struktur (Specflow, o. D.).

Akzeptanztest-getriebene Entwicklung (ATDD)

Bei der Methode der akzeptanztest-getriebenen Entwicklung (ATDD) wird ein einzelner Test aus der Perspektive des:der Kund:in und mit dem Fokus auf Funktionalität entwickelt. ATDD und BDD sind sich sehr ähnlich. Der Hauptunterschied ist der Schwerpunkt: Bei ATDD liegt er auf der Funktionalität, und bei BDD liegt er eher auf dem Verhalten. Bei dieser Entwicklungsmethode nehmen verschiedene Projektstakeholder:innen mit verschiedenen Perspektiven an der Konzipierung von Akzeptanztests teil, bevor die relevante Funktionalität implementiert wird. Diese Akzeptanztests spiegeln die Perspektive des:der Benutzer:in wider. In der folgenden Tabelle werden die wichtigsten Unterschiede der drei Methoden, TDD, BDD und ATDD, aufgeführt (Unadkat, 2020).

Tabelle 2: TDD, BDD und ATDD im Vergleich

	TDD	BDD	ATDD
Definition	Entwicklungsansatz zur Implementierung einer Funktion	Entwicklungsansatz auf der Grundlage des Systemverhaltens	Entwicklungsansatz für die Erfassung von Anforderungen (ähnlich wie beim BDD) durch Akzeptanztests, die vor der Implementierung der relevanten Funktionalität geschrieben werden
Hauptfokus	Modultests	Anforderungen aufgrund des Systemverhaltens verstehen	Schreiben von funktionalen Akzeptanztests
Teilnehmende	Entwickler:innen	Entwickler:innen, Kundschaft, Qualitätsicherungsingenieur:innen	Entwickler:innen, Kund:innen, Qualitätssicherungsingenieur:innen
Sprache	ähnlich wie der eigentliche Code	natürliche Sprache	natürliche Sprache

Quelle: Alvarid (2020), in Anlehnung an Unadkat (2020).

Am Ende dieses Abschnitts gehen wir noch kurz auf die Paarprogrammierung ein. Dabei handelt es sich um eine agile Softwareentwicklungstechnik, bei der zwei Programmierer:innen an einer Workstation arbeiten. Eine:r der beiden ist der:die sogenannte Pilot:in (englisch „driver“) und schreibt den Code, während der:die andere, als sogenannte:r Navigator:in (english „navigator“), den Code prüft und gleichzeitig die Übersicht behält. Es wurde festgestellt, dass bei der Paarprogrammierung zwar die Entwicklungszeit um ca. 15 % anstieg, jedoch die Qualität des Designs, die technischen Fähigkeiten und die Teamkommunikation verbessert und gleichzeitig Defekte und Personalrisiken gemindert wurden (Cockburn & Williams, 2001). Die Paarprogrammierung kam im Zuge des Extreme Programmings auf (Beck, 1999). Extreme Programming ist eine Softwaretechnik, die auf Informationsaustausch, Klarheit, Rücksicht, Einsatzschnelligkeit und -willigkeit beruht. Das Ziel des Extreme Programming ist es, tiefgehende Programmiererfahrung zu sammeln (Unadkat, 2020).

3.3 Kontinuierliche Integration und kontinuierliche Lieferung

In diesem Abschnitt befassen wir uns mit den Verfahren und der Automatisierung von kontinuierlicher Integration (CI), Lieferung (CD) und Training (CT) von ML-Modellen. Dazu müssen die MLOps-Prinzipien auf ein Machine-Learning-Projekt angewandt werden. MLOps kommt vom englischen Begriff Machine Learning Operations und ist ein ML-Entwicklungsansatz, der darauf abzielt, ML-Systementwicklung (daher ML) und -Betrieb (also Ops) zu kombinieren. Der Einsatz von MLOps beruht auf den Schwerpunkten Automation und Monitoring in allen Phasen des ML-Systemdesigns, also bei der Integration, beim Testen, dem Release, der Bereitstellung und im Infrastrukturmanagement (Google, 2020b). Befassen wir uns zunächst mit der kontinuierlichen Integration, der kontinuierlichen Lieferung und dem kontinuierlichen Testen.

Kontinuierliche Integration

Die kontinuierliche Integration (CI) ist eine Entwicklungsphilosophie, bei der Entwickler:innen in regelmäßigen Abständen Code in einem Versionsverwaltungs-Repository durch den sogenannten Commit ablegen oder committen; dies passiert normalerweise mindestens einmal pro Tag. Ein Anlass dafür ist, dass dadurch Fehler und andere Softwarequalitätsprobleme leichter und an kleineren Codestücken gefunden werden können. Dies ist an großen, über längere Zeiträume hinweg geschriebenen Codestücken deutlich schwieriger. Die kontinuierliche Integration geht auf Kent Beck zurück (Beck, 1999). Wie bei anderen Praktiken des Extreme Programming hatte man sich auch bei der kontinuierlichen Integration gefragt, warum man nicht ständig integrieren sollte, nachdem die regelmäßige Integration der Codebasis schließlich funktionierte. Vor der Implementierung von CI in Softwareprojekten müssen folgende Komponenten eingerichtet werden (Humble & Farley, 2015):

- **Versionsverwaltung:** Alle Elemente eines Softwareprojekts sollten in einem Versionsverwaltungs-Repository wie Git abgelegt werden (Git, o. D.). Zu diesen Elementen gehören der Quellcode, die Testskripts, die Datenbankskripts, die Build- und Bereitstellungsskripts, die Konfigurationsdateien usw.
- **Ein automatisierter Build:** Der Build des Programms sollte in automatisierter Form über eine Befehlszeile laufen können. Zum Beispiel könnte einer integrierten Entwicklungsumgebung (IDE) über ein Befehlszeilenskript der Befehl zum Erstellen der Software gegeben werden. Es könnte aber auch eine komplexe Kombination aus mehrstufigen Build-Skripts, die einander aufrufen, sein.
- **Teamvereinbarung:** Kontinuierliche Integration ist kein Tool sondern Teamarbeit; daher müssen die Mitglieder eines Entwicklungsteams Bereitschaft und Disziplin aufweisen. Erst nachdem sich die Teammitglieder der notwendigen Disziplin bewusst sind, kann kontinuierliche Integration zu einer Verbesserung der Produktqualität führen.

CI kann zum Beispiel von folgenden Tools unterstützt werden: GoCD (GoCD, o. D.), CruiseControl (CruiseControl, o. D.), and Jenkins (Jenkins, o. D.-a). Nach der Installierung des CI-Tools kann der CI-Prozess mit der Konfigurierung des Tools beginnen. Bei der Konfigurierung wird der Ort des Quellcode-Repositories, die für die Ausführung der Softwarekomplikierung notwendigen Skripts und die automatisierten Commit-Tests festgelegt.

Kontinuierliches Testen

Wie bereits im ersten Abschnitt dieser Lektion erwähnt wurde, unterstützt das automatisierte Testen die Testingenieur:innen während des Softwareentwicklungs-Lebenszyklus beim Schreiben und Ausführen verschiedener Testarten. Die Testarten reichen von einem Modultest zum System- und Regressionstest der gesamten Software (Sakolick, 2020). Beim Regressionstest werden funktionale und nicht-funktionale Tests noch einmal an der bereits getesteten Software ausgeführt, um zu prüfen, ob die nun modifizierte Software den Test nach der Integration des geänderten Codes besteht (Basu, 2015). Regressionstests werden nach anderen automatisierten Tests, z. B. Performanztests, API-Tests und Sicherheitstests, vorgenommen. Alle genannten Tests müssen durch eine Befehlszeile oder andere Automationstools ausgelöst werden. Nach der Automatisierung des Testverfahrens bedeutet kontinuierliches Testen (Cte), dass ein automatisierter Test bereits in der CI/CD-Pipeline vorhanden ist. In diesem Fall sollten einige Tests (z. B. Modul- und Funktionalitätstests) bereits in den CI-Teil der Pipeline integriert worden sein (CI weist dann auf Probleme vor oder während der Integration hin).

Tests, für die Lieferungskonditionen vollständig sein müssen, z. B. Performanz- oder Sicherheitstests, sollten in die kontinuierliche Lieferung integriert werden. Diese Tests werden nach dem Build ausgeführt (Sakolick, 2020).

Kontinuierliche Lieferung

Die kontinuierliche Lieferung (CD) ist eine Softwareentwicklungsmethode, bei der Projektteams ständig wertige Software in kurzen Zyklen liefern und sicherstellen, dass die Software jederzeit zuverlässig geliefert werden kann (Chen, 2015). Durch die kontinuierliche Lieferung können Unternehmen Dienstverbesserungen schnell, effizient und zuverlässig

auf den Markt bringen. Während der Entwicklungsphase verwendet jedes Entwicklungsteam eine oder mehrere Entwicklungs- und Testumgebungen zur Bereitstellung von Anwendungsänderungen zum Testen. Dieser Prozess kann mithilfe von CI/CD-Tools wie Jenkins (Jenkins, o. D.-c), CircleCI (CircleCI, o. D.) oder Travis CI (Travis CI, o. D.) automatisiert werden.

Eine typische CD-Pipeline beinhaltet die folgenden Schritte (Chen, 2015):

1. **Commit ausführen:** Beim Commit bekommen Entwickler:innen schnelles Feedback zum bereits von ihnen eingecheckten Code. Wenn ein:e Entwickler:in den Code ins CI/CD-Tool eincheckt, wird dieser Schritt automatisch ausgelöst und der Quellcode und die Modultests werden ausgeführt. Tritt ein Fehler auf, wird der Vorgang in der Pipeline abgebrochen und die Entwickler:innen erhalten eine Fehlermeldung. Der:Die Entwickler:in klärt das Problem und checkt den Code ein. Nach erneuter und fehlerfreier Ausführung des Codes erfolgt der nächste Schritt in der Pipeline.
2. **Build:** Bei diesem Schritt werden Modultests zur erneuten Erstellung von **Code Coverage**-Berichten durchgeführt. Dabei werden Integrationstests und verschiedene Codeanalysen ausgeführt. Als Output werden Artefakte generiert. Diese werden ins Repository geladen, in dem sie für die Bereitstellung und die Lieferung verwaltet werden. Alle der nachfolgenden Pipeline-Schritte werden mithilfe dieser Artefakte ausgeführt.
3. **Akzeptanztest:** Durch diesen Schritt wird gewährleistet, dass die Software alle definierten Benutzer:innenanforderungen erfüllt. Dabei wird von der Pipeline eine Akzeptanztestumgebung geschaffen, d. h. eine der Produktion gleichen Umgebung, in der die Software bereitgestellt wird. Bei diesem Schritt müssen die Server bereitgestellt und konfiguriert sowie die Software auf den Servern bereitgestellt und konfiguriert werden. Wenn die Software den Akzeptanztest in dieser Umgebung besteht, beginnt der nächste Schritt in der Pipeline.
4. **Performanztest:** Bei diesem Schritt beurteilt die Pipeline, inwiefern Codeänderungen die Leistung der Software beeinflusst. Die Pipeline führt eine Reihe von Performanztests durch und erstellt nach deren Ausführung entsprechende Berichte.
5. **Produktion:** Der letzte Schritt ist die Bereitstellung in der Produktionsumgebung.

Die genannten Schritte können mithilfe von CI/CD-Tools, wie Jenkins, automatisiert werden. In diesem Fall könnten Entwickler:innen Jenkins zur Beschreibung der Pipeline in einer Jenkinsfile, die verschiedene Stufen enthält, verwenden. Die Jenkinsfile enthält zudem Umgebungsvariablen, Zertifikate, und andere Parameter (Jenkins, o. D.-c).

Abbildung 29: CD-Pipeline



Quelle: Alvarid (2020).

Kontinuierliches Training

Kontinuierliches Training (CT) wird durch die Automatisierung der ML-Pipeline erreicht. Um die ML-Pipeline durch die Versorgung des Modells der Produktionsumgebung mit neuen Daten zu automatisieren, müssen folgende Schritte in der ML-Pipeline implementiert werden (Google, 2020b).

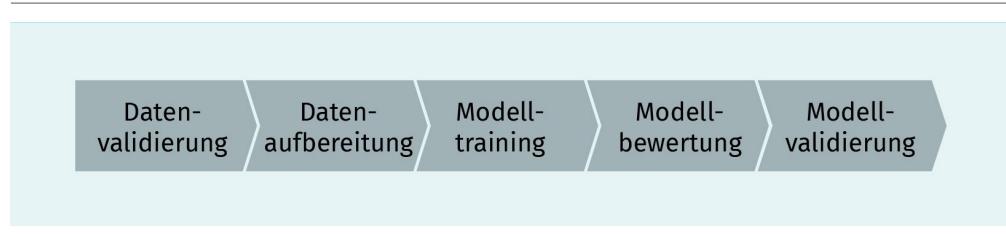
- **Automatische Datenvalidierung:** Dieser Schritt wird vor dem Training des Modells ausgeführt, um zu prüfen, ob das Modell trainiert oder die Pipeline-Ausführung angehalten werden sollte.
- **Automatische Modellvalidierung:** Dieser Schritt wird mithilfe der neuen Daten unmittelbar nach dem Modelltraining durchgeführt. Er ist zur Validierung des Modells vor der Produktion notwendig.
- **Pipeline-Auslöser:** Die Pipeline könnte zur Aufnahme neuer Daten nach bedarfsbasierter, manueller Ausführung der Pipeline, nach einem Terminplan (täglich, wöchentlich usw.), bei Verfügbarkeit neuer Daten, bei einem Abfall der Modellleistung oder bei erheblichen Änderungen der Datenverteilung ausgelöst werden.
- **Metadatenverwaltung:** Nach jeder Ausführung der Pipeline sollten Metadaten — wie Pipeline- und Komponentenversionen, Datum, Anfangs- und Endzeit der Ausführung, der Ausführende oder die an die Pipeline abgegebenen, konfigurierten Parameter — im Metadatenspeicher der Pipeline gespeichert werden.

Anhang 1 des vorliegenden Studienskripts enthält eine in Google Cloud dargestellte schematische Abbildung einer automatisierten ML-Pipeline für kontinuierliches Training (CT) (Google, 2020b).

Nachfolgend werden Eigenschaften einer ML-Pipeline für kontinuierliches Training aufgelistet (Google, 2020b):

- **Schnelle Experimente:** Der Übergang zwischen den Experimentierstufen in der Pipeline (Datenvalidierung, Datenaufbereitung, Modelltraining, Modellbewertung und Modellvalidierung) sollten so arrangiert werden, dass die Experimente in schneller Iteration stattfinden können.
- **Kontinuierliches Training des Modells in der Produktion:** Das Training des ML-Modells in der Produktion sollte automatisiert und mit aktuellen Daten durchgeführt werden.
- **Symmetrie zwischen Experiment und Betrieb:** Es ist ein entscheidender Aspekt einer automatisierten Pipeline, dass dieselbe Pipeline aus der Entwicklungs- oder Experimentumgebung in der Vorproduktions- und Produktionsumgebung verwendet wird.
- **Pipeline-Komponenten in Codemodulen:** Die Komponenten der ML-Pipelines müssen wiederverwendbar, zusammenstellbar und evtl. auch über ML-Pipelines hinweg teilbar sein. Daher muss der Quellcode für Komponenten in Modulen erstellt werden.

Abbildung 30: Orchestrierte Experimente



Quelle: Alvarid (2020).

MLOps und ML-Pipeline für CI/CD

Ein ML-System ist auch ein Software-System, daher müssen ähnliche Vorgehensweisen angewendet werden, um zu gewährleisten, dass im großen Stil Builds erstellt und die ML-Systeme zuverlässig ausgeführt werden können. Es gibt jedoch eine Reihe von Unterschieden zwischen ML-Systemen und konventionellen Softwaresystemen (Google, 2020b). Beispielsweise sollten einem ML-Projektteam Datenwissenschaftler:innen angehören, die Erfahrungen in explorativer Datenanalyse, Modellentwicklung und Experimenten mitbringen. Das bedeutet jedoch nicht, dass sie gleichzeitig über Softwareentwicklungsskills verfügen. Auch der Entwicklungsprozess unterscheidet sich, denn die ML-Systementwicklung basiert auf Experimenten. Das heißt, dass Entwickler:innen verschiedene Algorithmen, Modellierungstechniken und Parameter ausprobieren sollten, um die beste Lösung für ein Problem zu finden. Neben konventionellen Tests — wie Modul- und Integrationstests — sollten auch die Daten- und Qualitätsvalidierung der trainierten Modelle getestet werden. Zudem kann die Bereitstellung der ML-Systeme abweichen, da in manchen Fällen eine mehrstufige Pipeline notwendig ist, wodurch die Bereitstellung komplexer wird. Da sich ML-Systeme ständig und sehr dynamisch weiterentwickeln, ist die Wahrscheinlichkeit der Leistungsminderung eines Modells höher als bei gewöhnlichen Softwaresystemen.

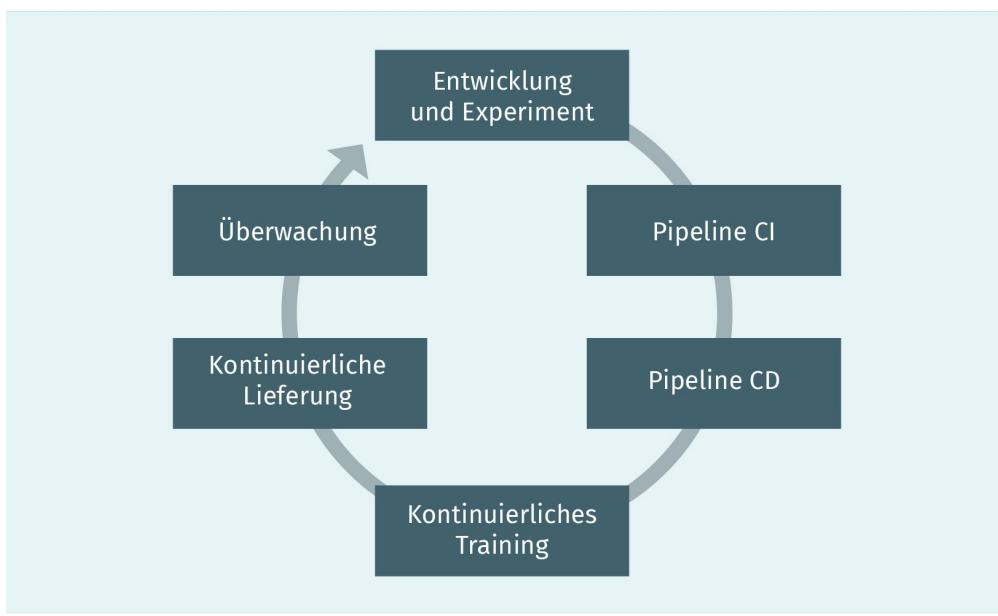
Angesichts der oben angeführten Eigenschaften von ML-Systemen bestehen auch im Rahmen der kontinuierlichen Integration und Lieferung Unterschiede zu konventionellen Softwaresystemen (Google, 2020b):

- Zusätzlich zum Code und zu den Modulen müssen bei der kontinuierlichen Integration von ML-Systemen auch Daten, Datenschemata und Modelle getestet und validiert werden.
- Neben einer Software oder einem Dienst sollten bei der kontinuierlichen Bereitstellung von ML-Systemen in der ML-Trainingspipeline automatisch auch andere Dienste (z. B. Modellvorhersage) bereitgestellt werden.
- Das kontinuierliche Training ist eine neue Funktion und trifft nur auf ML-Systeme zu. Dabei geht es in erster Linie um das automatische Training und die Pflege der Modelle.

Hinsichtlich der genannten Unterschiede besteht die CI/CD-Automation von ML-Systemen aus den sechs Schritten, die in der nachstehenden Abbildung aufgeführt sind (Google, 2020b):

- Entwicklung und Experiment:** Durch die Untersuchung verschiedener ML-Algorithmen und neuer ML-Modelle kann das Entwicklungsteam den besten Quellcode für die ML-Pipeline bestimmen. Dieser wird ins Quell-Repository als Output dieses Schritts gepusht.
- Kontinuierliche Integration der Pipeline:** Der Quellcode aus Schritt 1 wird ausgeführt und getestet. Das Output aus diesem Schritt sind mehrere Pipelinekomponenten, z. B. Pakete, auszuführende Dateien und Artefakte, die in den nächsten Schritten verwendet werden.
- Kontinuierliche Lieferung der Pipeline:** Die Artefakte aus Schritt 2 werden in der Zielumgebung bereitgestellt. Als Output aus diesem Schritt resultiert eine bereitgestellte Pipeline mit neuem ML-Modell.
- Automatisches Auslösen:** Die Pipeline wird automatisch in der Produktionsumgebung gestartet (als Reaktion auf ein Signal oder auf Basis eines Zeitplans). Das Output aus diesem Schritt ist ein programmiertes Modell, das in die Modell-Registrierung gepusht wird.
- Kontinuierliche Lieferung des Modells:** Das trainierte Modell wird als Vorhersagemodell an den Kund:in geliefert. Aus diesem Schritt resultiert der Vorhersagedienst.
- Monitoring:** Statistische Daten zur Modellperformance werden im aktiven System gesammelt. Aus diesem Schritt geht neben einem Bericht möglicherweise ein Signal zur Auslösung von Schritt 1 in der Pipeline hervor.

Abbildung 31: CI/CD für ein ML-Modell



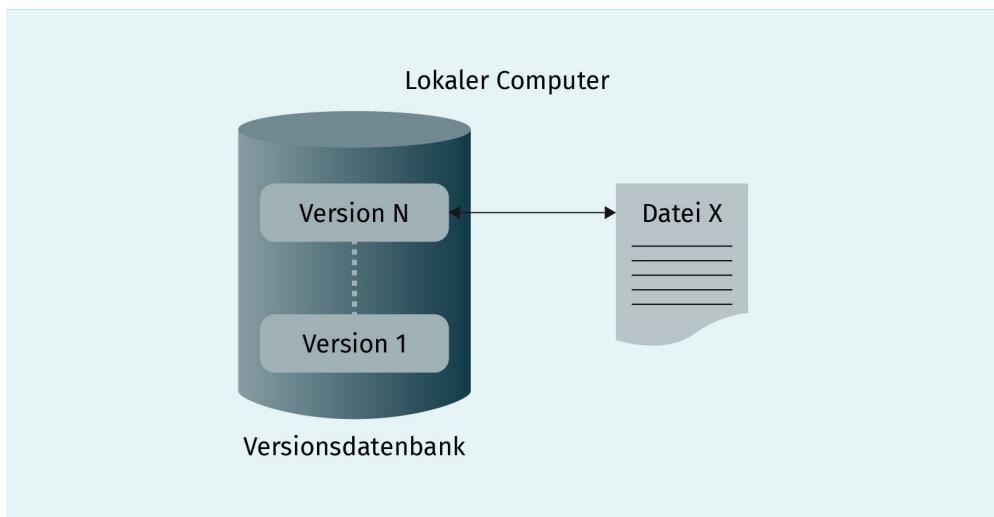
Quelle: Alvarid (2020).

3.4 Versionsverwaltung

Während des Softwareentwicklungszyklus muss das Entwicklungsteam alle Änderungen an den Anwendungen, wie den Quellcode, die Projektdokumente oder Buildskripts, aufzeichnen. Dies ist vor allem bei großen Softwareprojekten mit vielen Entwickler:innen und verschiedenen Teams wichtig.

Daher wird seit Jahrzehnten an einer Lösung gearbeitet: einem Versionsverwaltungssystem (VCS). Mithilfe eines VCS und indem ein Dokumentationssystem geführt wird, können mehrere Entwickler:innen und Teams gemeinsam an einem Projekt und parallel an separaten Teilen einer Anwendung arbeiten (Humble & Farley, 2015). Außerdem können sowohl ausgewählte Dateien als auch das gesamte Projekt auf einen früheren (gespeicherten) Zustand zurückgesetzt werden. Das erste Versionsverwaltungssystem, SCCS (Source Code Control System), wurde im Jahr 1972 von Marc J. Rochkind bei Bell Labs entwickelt (Rochkind, 1975). Dem folgten weitere Open-Source-Versionsverwaltungssysteme wie z. B. RCS (Tichy, 1982), CVS (Price, 2005), Apache Subversion (Free Software Directory, o. D.) und Git (Git, o. D.) sowie einige kommerzielle Lösungen wie Perforce (Perforce, o. D.), StarTeam (StarTeam, o. D.), IBM Rational ClearCase (IBM, o. D.-b), Mercurial (Mercurial, o. D.) und das Microsoft Teams Foundation System (MS Teams, o. D.). Dieser Abschnitt thematisiert eines der Open-Source-Versionsverwaltungssysteme. Da SCCS- und RCS-Lösungen heutzutage nicht mehr sehr gebräuchlich sind, werden diese nicht weiter behandelt. Stattdessen wird Git, das am häufigsten eingesetzte Versionsverwaltungssystem, beleuchtet. Zuvor soll ein kurzer Blick auf die verschiedenen VCS-Kategorien geworfen werden. Beginnen wir mit den lokalen Versionsverwaltungssystemen. Diese Kategorie ist die einfachste Form eines VCS-Systems, die wahrscheinlich durch die meisten Entwickler:innen beim anfänglichen Kodieren verwendet wurde. Dabei werden modifizierte Dateien in ein neues, mit Datum (und bestenfalls auch mit Zeitstempel) versehenes Verzeichnis kopiert. Für eine:n einzelne:n Entwickler:in stellt das einen einfachen Ansatz dar, jedoch ist dieser sehr anfällig für Fehler wie den Verlust der Verzeichnisstruktur und das Überschreiben von Dateien. Um dieses Problem zu verhindern, kann eine einfache lokale Versionsdatenbank entwickelt werden, durch die Änderungen nachverfolgt werden.

Abbildung 32: Lokales Versionsverwaltungssystem



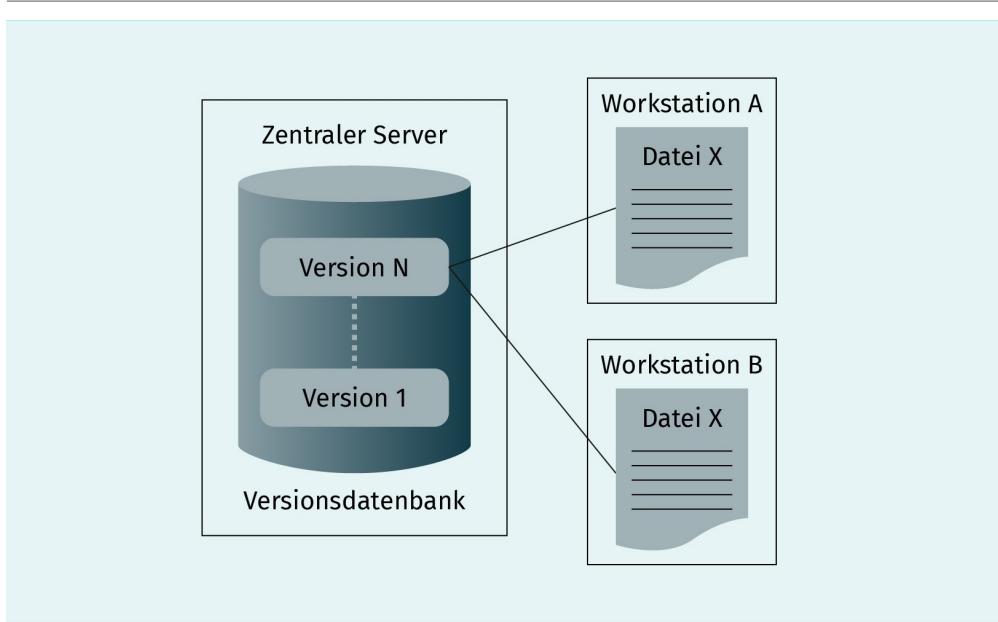
Quelle: Alvarid (2020), in Anlehnung an Git (2020).

Die zweite Kategorie sind die zentralen Versionsverwaltungssysteme (CVCS). Ist mehr als eine Person oder Arbeitsgruppe am Projekt beteiligt, sollte eine andere VCS-Lösung verwendet werden, wie es die nächste Abbildung zeigt. In einem CVCS werden Informationen über Dateiänderungen auf einem Zentralserver gespeichert und mit Clients (Workstations) geteilt. Zu den verfügbaren Produkten dieser Kategorie gehören Subversion (Free Software Directory, o. D.), Perforce (Perforce, o. D.) und CVS (Price, 2005).

Dieses System hat z. B. den Vorteil, dass alle berechtigten Teammitglieder sehen können, woran die anderen arbeiten. Außerdem können durch die Administrator:innen Rollen festgelegt werden. Ein Nachteil besteht beispielsweise im **Single Point of Failure**; so kann ein Server bei einem Fehler die Workstations so beeinträchtigen, dass Änderungen nicht mehr nachverfolgt werden können.

Single Point of Failure
Ein nicht-redundanter Teil eines Systems, dessen Fehlfunktion das gesamte System zum Erliegen bringt, wird als SPOF oder Single Point of Failure bezeichnet (AVI Networks, o. D.).

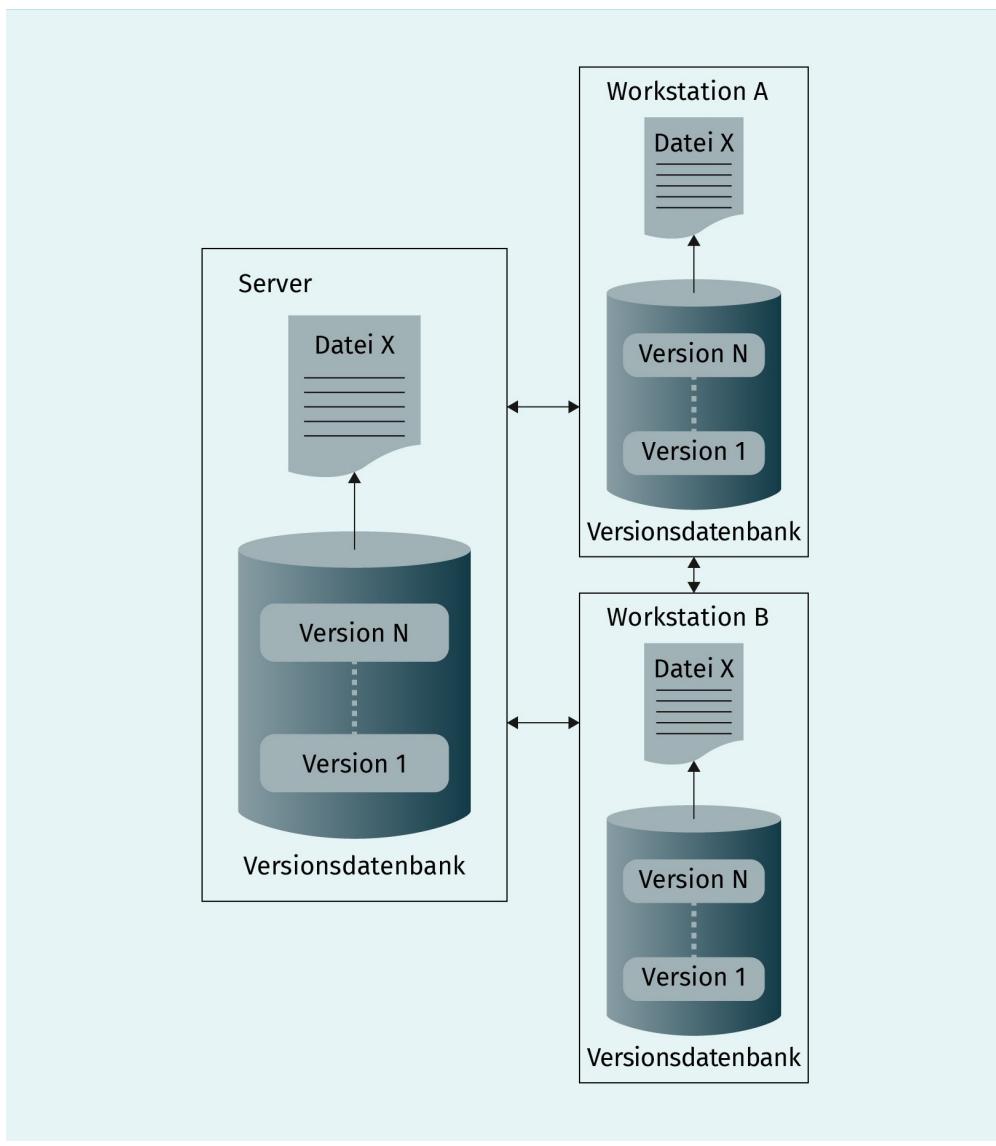
Abbildung 33: Zentrales Versionsverwaltungssystem



Quelle: Alvarid (2020), in Anlehnung an Git (2020).

Die beste Lösung zur Vermeidung von Problemen mit lokalen und zentralen VCS ist ein verteiltes Versionsverwaltungssystem (DVCS) wie Git (Git, o. D.) oder Mercurial (Mercurial, o. D.). In einem DVCS überprüfen Entwickler:innen die letzte Dateiversion nicht, aber sie erstellen eine Spiegelkopie des Repository. Somit wird das Single Point of Failure-Problem gelöst, da beim DVCS das Repository eines jeden Benutzers auf den Zentralserver zur Wiederherstellung der verlorenen Daten kopiert werden kann.

Abbildung 34: Verteiltes Versionsverwaltungssystem



Quelle: Alvarid (2020), in Anlehnung an Git (2020).

Git

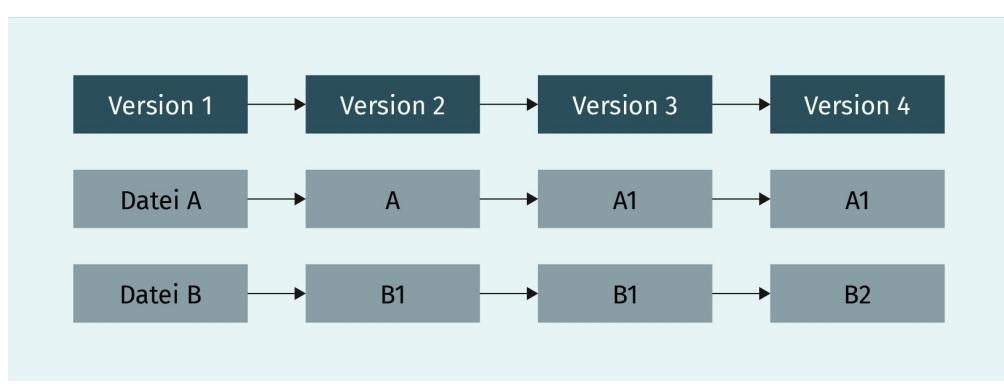
Die Geschichte des Git-Projekts ist eng mit dem Open-Source-Softwareprojekt Linux Kernel verbunden. Im Jahr 2005 entschied sich das Linux Kernel-Team dazu, das verteilte Versionsverwaltungssystem BitKeeper nicht weiterzuführen und ein eigenes Versionsverwaltungssystem zu entwickeln. Aus dieser Entscheidung ging das Git-Projekt hervor (BitKeeper, o. D.). Git ist ein Versionsverwaltungssystem, das einfach aber schnell ist und in besonderem Maße die nicht-lineare Entwicklung unterstützt. Git verfügt über Tausende voll verteilte **Parallelzweige**, mit denen große Projekte verwaltet werden können. Während andere VCS die Änderungsinformationen als Liste von dateibasierten Änderungen

Entwicklungszweig
Eine Kopie eines Haupt-Repository des Versionsverwaltungssystems.

speichern (d. h. als Dateisatz mit den entsprechenden Änderungen), sichert Git eine Momentaufnahme jedes Projektzustands unmittelbar nach einem Commit, bei dem Änderungen auf dem lokalen Repository durch den:die Benutzer:in gespeichert werden.

Zur Effizienzverbesserung wird in der Momentaufnahme nur eine Verknüpfung zum vorherigen Zustand der Datei angeboten, wenn sich die Datei nicht geändert hat. In diesem Sinne könnte Git als Minidateisystem betrachtet werden (Git, o. D.).

Abbildung 35: Momentaufnahmen in Git als Versionsverwaltung



Quelle: Alvarid (2020), in Anlehnung an Git (2020).

Ein Vorteil von Git ist die Prüfsumme. Für jede Änderung wird eine Prüfsumme berechnet, bevor sie gespeichert wird. Änderungen können nicht vorgenommen werden, ohne dass Git darüber informiert wird. Durch diese Funktion wird verhindert, dass Informationen während der Übertragung verloren gehen. Um diese Prüfsummen zu erstellen, verwendet Git das SHA-1-Hash, eine 40-Zeichen umfassende Folge aus hexadezimalen Zeichen (Git, o. D.). Dieses Hash wird aus dem Inhalt der Datei oder der Verzeichnisstruktur in Git berechnet.

Eine Datei in Git kann drei verschiedene Zustände haben:

1. **Geändert:** Die Datei wurde geändert, aber die Änderungen wurden noch nicht in der Datenbank gespeichert. In diesem Zustand wird die Datei mit `modified` gekennzeichnet.
2. **Für Commit vorgemerkt:** Eine geänderte Datei wurde von einem: einer Benutzer:in kennzeichnet, so dass sie bei der nächsten Commit-Momentaufnahme mitverarbeitet wird. In diesem Zustand wird die Datei mit `staged` gekennzeichnet.
3. **Committed:** Die Daten wurden sicher in der lokalen Datenbank gespeichert. In diesem Zustand wird die Datei mit `committed` gekennzeichnet.

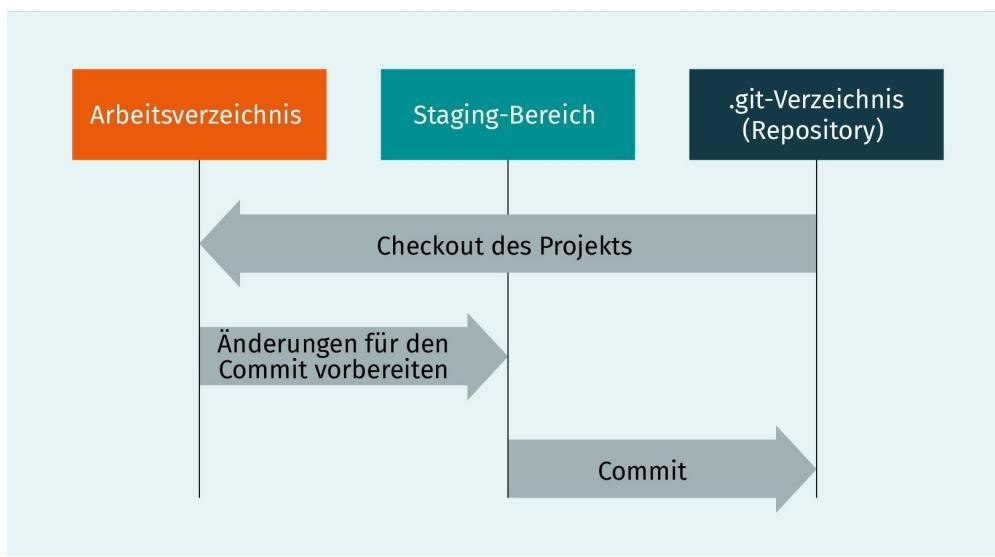
Basierend auf diesen drei Zuständen gibt es in einem Git-Projekt drei Hauptabschnitte oder -stufen. Der erste Abschnitt ist das Arbeitsverzeichnis, ein einzelner **Checkout** einer Projektversion. Die Dateien in diesem Verzeichnis werden aus dem Git-Repository entnommen, wobei Änderungen aus dem Remote-Repository in den aktuellen Zweig integriert werden. Der zweite Abschnitt ist der Staging-Bereich. Dabei handelt es sich um eine Datei im Git-Verzeichnis, die Informationen zu den Änderungen enthält, die im nächsten

Commit integriert werden. Die dritte und letzte ist das Git-Verzeichnis selbst. Es ist der wichtigste Teil von Git, in dem die Metadaten- und Objektdatenbanken des Projekts gespeichert werden. Wenn ein:e Benutzer:in ein Repository aus einer anderen Workstation klonen, heißt das, dass das Git-Repository kopiert wird. Dieser Prozess wird in der folgenden Abbildung gezeigt. Werden die Abschnitte eines Git-Projekts betrachtet, ergeben sich bei einem einfachen Git-Workflow die folgenden Schritte (Git, o. D.):

1. Der:Die Benutzer:in ändert die Datei im Arbeitsverzeichnis, was von Git nachverfolgt wird.
2. Dann stellt er:sie Teile der Änderungen bereit, die im nächsten Commit im Staging-Bereich gespeichert werden sollen.
3. Der letzte Schritt ist der Commit selbst. Dabei wird eine Momentaufnahme der Dateien im Staging-Bereich festgehalten und im Git-Directory gespeichert.

Checkout
Beim Auschecken aus dem Entwicklungszweig werden die Dateien im Arbeitsverzeichnis aktualisiert, sodass sie der Version im Entwicklungszweig entsprechen. Dabei wird Git angewiesen, alle neuen Commits auf diesem Entwicklungszweig zu speichern (Atlassian Bitbucket, o. D.).

Abbildung 36: Abschnitte in einem Git-Projekt

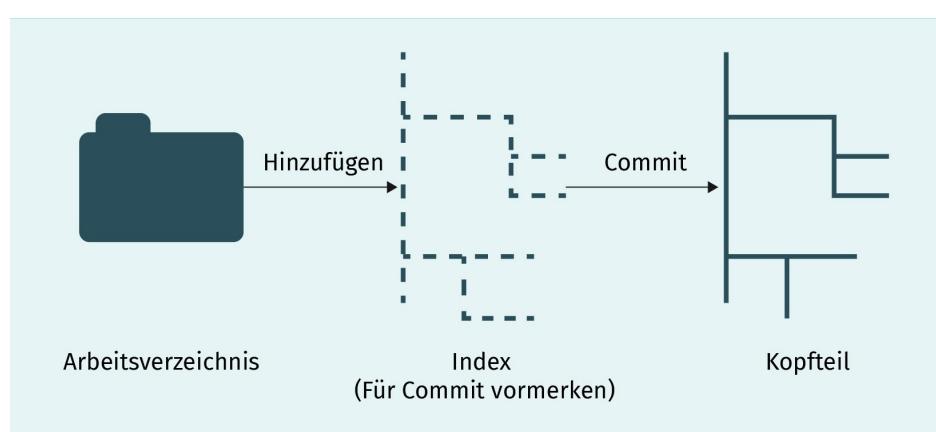


Quelle: Git (2020).

Nachstehend wird ein praktisches Beispiel eines Git-Workflows gezeigt (Dudler, o. D.):

1. Ein neues Repository im Projektverzeichnis erstellen.
`git init`
2. Eine Arbeitskopie auf einem lokalen Repository durch die Ausführung des Befehls erstellen.
`git clone /path/to/repository`
3. Das lokale Repository besteht aus drei Baumstrukturen:
 - a) Das Working Directory (Arbeitsverzeichnis) enthält die eigentlichen Dateien.
 - b) Index ist der Staging-Bereich.
 - c) HEAD zeigt auf die letzte Commit-Version.

Abbildung 37: Lokale Repository-Bäume für Git



Quelle: Alvarid (2020).

4. Änderungen vorschlagen (dem Index hinzufügen).
`git add <filename>`
5. Commit der Änderungen in den Head durchführen.
`git commit -m "Commit message"`
6. Änderungen ins Remote-Repository schicken.
`git push origin master`
7. Einen neuen Entwicklungszweig („branch_1“) erstellen.
`git checkout -b branch _1`
8. Switch zum Hauptentwicklungszweig, dem Master, vornehmen.
`Git checkout master`
9. Push des Entwicklungszweig ins Remote-Repository durchführen.
`Git push origin <branch>`
10. Das lokale Repository mit dem letzten Commit aktualisieren.
`Git pull`

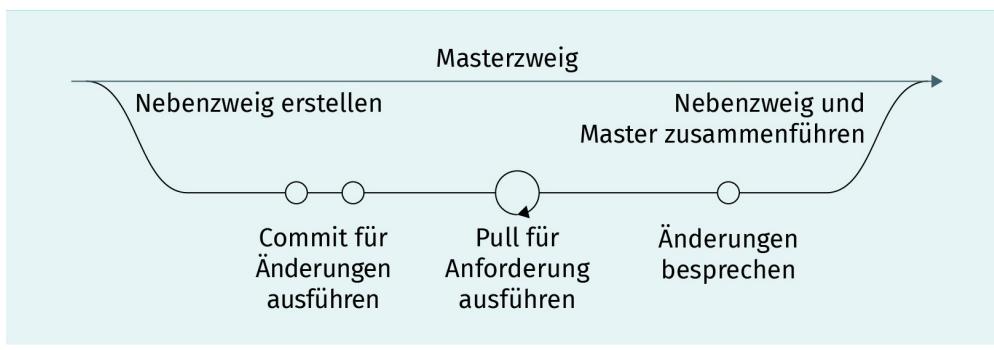
GitHub

GitHub ist eine cloudbasierte Entwicklerplattform auf der Grundlage von Git (GitHub, o. D.-a). Der Unterschied zwischen Git und GitHub ist, dass Git ein Open-Source-Tool ist, mit dem Entwickler:innen ihren Quellcode lokal verwalten können, während GitHub eine cloudbasierte Plattform ist, auf der Entwickler:innen ihre Projekte mit der Community teilen können, durch die sie Zugriff auf Entwicklungstools haben, wo sie gegenseitig Code überprüfen können und vieles mehr. Wie bei Git liegt auch einem GitHub-Projekt ein Projekt-Repository zugrunde. Ein Projekt-Repository enthält alle Komponenten, die für ein Projekt benötigt werden: Dateien, Ordner, Tabellenkalkulationen, Datensätze etc. Normalerweise ist darin auch eine README-Datei enthalten, die Informationen über das Projekt enthält.

Standardmäßig hat ein Repository einen Entwicklungszweig, den `master`. Bei der Verzweigung wird gleichzeitig an verschiedenen Versionen des Repositories gearbeitet. Die Entwicklungszweige werden dazu verwendet, die Änderungen in den Quelldateien zu tes-

ten, bevor ein Commit in den master stattfindet. Wenn Entwickler:innen in einem Entwicklungszweig des master arbeiten und im master Änderungen entstehen, könnten sie die Änderungen aus dem master in den Entwicklungszweig holen.

Abbildung 38: Verzweigung und GitHub



Quelle: Alvarid (2020), in Anlehnung an Git (2020).

3.5 Entwicklertools

Programmier- oder Softwareentwicklertools sind Computerprogramme, die Softwareentwickler:innen und Entwicklungsteams bei der Umsetzung von Softwareprojekten unterstützen. Diese Werkzeuge werden dazu verwendet, Quellcode für Programme mit Texteditoren zu erstellen und zu modifizieren, den Programmfluss mithilfe von speziellen Editoren mit grafischen Benutzeroberflächen zu unterstützen, den Quellcode mit Compilern und Assemblern in ausführbare Maschinensprache zu übersetzen, entwickelte Lösungen mit Testtools oder Debuggern zu testen und zu debuggen, und Programme und die zugehörige Dokumentation anhand von Versionsverwaltungssystemen zu speichern und zu verwalten. In diesem Abschnitt betrachten wir einige der gebräuchlichsten Entwicklertools.

Befehlszeilschnittstelle

Das einfachste und geläufigste Entwicklertool ist die Befehlszeilschnittstelle. Sie ist Teil eines Computerprogramms, das eine Textzeile als Input vom Benutzer erhält und diese mithilfe des Befehlszeileninterpreters in den Kontext des entsprechenden Betriebssystems bzw. in die Programmiersprache übersetzt (Kumar, 2016). Betriebssysteme und Programmiersprachen implementieren die Befehlszeilschnittstelle in die **Shell**, um Zugriff auf das Betriebssystem zu erhalten oder Code zu schreiben. Als Beispiele seien Unix Shell (Bourne, 1978), PowerShell (Bright, 2016), Z Shell (Z Shell, o. D.) und Python Shell (Python Shell, 2020) aufzuführen.

Shell
Eine Software, bei der ein:e Benutzer:in mit dem Betriebssystem oder dem Programm selbst interagiert.

Ein Befehl in einer Betriebssystem-Befehlszeilschnittstelle hat z. B. folgende Komponenten:

```
prompt command parameter_1, ..., parameter_n
```

Die Eingabeaufforderung (prompt) gibt den Kontext für den:die Benutzer:in (und endet gewöhnlich mit einem der folgenden Zeichen: \$, %, #, :, > oder -).

Der Befehl (command) wird durch den:die Benutzer:in zur Ausführung einer bestimmten Aufgabe eingegeben. Parameters sind optionale Parameter vom Client, um den Befehl zu steuern oder einzuschränken. Obwohl eine Befehlszeilenschnittstelle der einfachste Weg ist, mit einem Betriebssystem zu kommunizieren oder Code zu schreiben, ist sie nicht immer die beste Lösung, vor allem nicht für Anfänger, zum Inline-Editing und zum Debuggen. Es gibt jedoch integrierte Tools für alle Befehlszeilenschnittstellen, die die Arbeit von Entwickler:innen erleichtern. Zu diesen Tools gehören der Texteditor Vim für Unix- und Apple OS-Befehlszeilenschnittstellen (Vim, o. D.), Wget zum Abrufen von Dateien anhand von HTTP, HTTPS, FTP und FTPS über das Internet (Free Software Foundation, o. D.-a) und das Datenkompressionsprogramm Gzip (Free Software Foundation, o. D.-b).

Integrierte Entwicklungsumgebung (IDE)

Eine integrierte Entwicklungsumgebung (IDE) ist ein einzelnes Programm oder eine Plattform, die verschiedene Entwicklungsbereiche zur Softwareentwicklung umfasst. Dazu gehören Microsoft Visual Studio, Eclipse, NetBeans und PyCharm. Zu den typischen Funktionen von modernen IDEs gehören:

- **Intelligente Code-Vervollständigung:** Bei dieser praktischen Methode wird die Softwareentwicklung durch Zugriff auf die Funktionsbeschreibungen und Funktionsparameterlisten beschleunigt. Nach der Eingabe eines Funktionsnamens erscheint eine kurze Funktionsbeschreibung und eine Liste von Inputparametern der Funktion. Ein Beispiel für derartige Funktionalität ist IntelliSense, das Code-Editorfunktionen, z. B. Code-Vervollständigung, Parameterinformationen, Schnellinformationen und Mitgliederlisten, miteinander kombiniert (Visual Studio Code, 2020).
- **Quellcode-Editor:** Der:Die Entwickler:in kann die IDE zum Schreiben und Editieren von Quellcode verwenden. Der Editiervorgang wird dabei von Funktionen, wie der intelligenten Code-Vervollständigung, unterstützt.
- **Build-Automatisierung:** Eine integrierte Entwicklungsumgebung kann den Build-Prozess automatisieren, d. h. Quellcode kompilieren, ein komprimiertes Format aus kompilierten Dateien erstellen und Installer fertigen.
- **Debugger:** Ein IDE-Debugger ermöglicht die Änderung von Variablenwerten und die Prüfung des Variablenwerts während der Ausführung, den Abbruch der Ausführung unabhängig vom Code usw.
- **Syntaxhervorhebung:** Eine IDE für die unterstützten Programmiersprachen hebt den Text in verschiedenen Farben und Schriftarten hervor, wie nachstehend abgebildet.

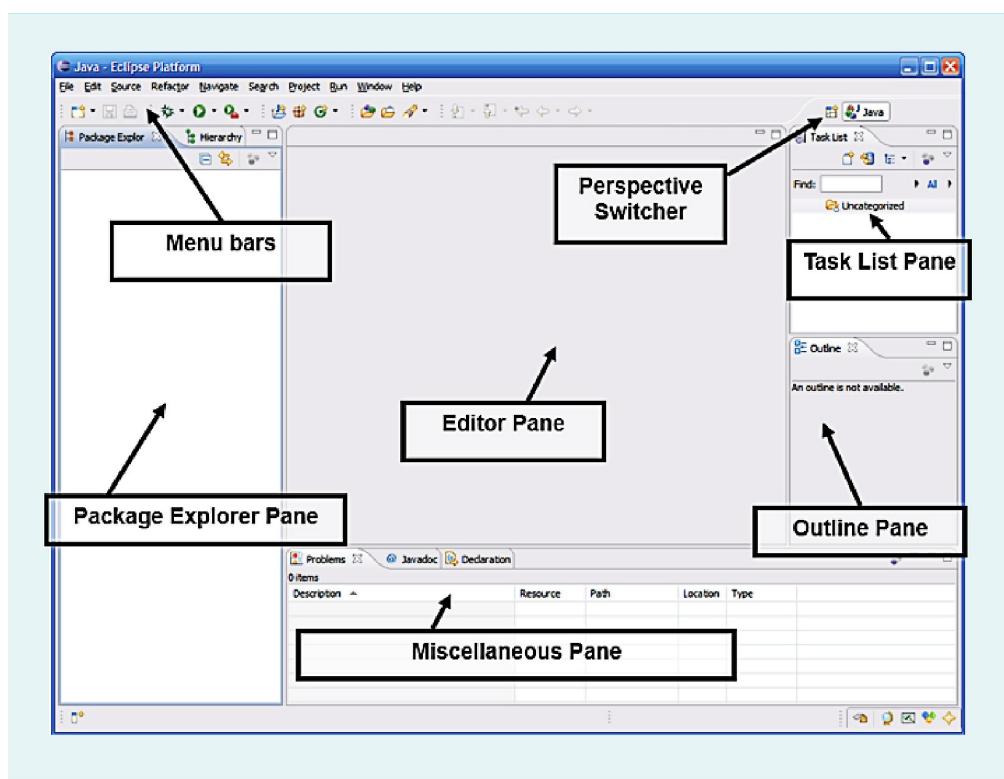
Abbildung 39: Syntaxhervorhebung in integrierter Entwicklungsumgebung

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import datetime
4 from datetime import datetime
5 import plotly as pltly
6 import plotly.express as px
7 import plotly.graph_objs as go
8 import plotly.io as pio
9 from PIL import Image
10
```

Quelle: Alvarid (2020).

Nun befassen wir uns mit der gängigsten Open-Source-IDE: Eclipse (Eclipse, o. D.). Eclipse begann als proprietäre Technologie unter der Führung von IBM. Im Jahr 2001 kündigte das Eclipse-Konsortium ein Open-Source-Projekt für Eclipse an (Eclipse, o. D.). Diese IDE stellt Tools zum Codieren, Ausführen und Debuggen von Anwendungen zur Verfügung und war ursprünglich für Java gedacht. Heute unterstützt Eclipse viele andere Sprachen, z. B. C, C++, Python und Ruby (Universität von Maryland, 2018). Die Komponenten der Eclipse-IDE werden nachstehend dargestellt.

Abbildung 40: IDE-Komponenten in Eclipse



Quelle: Alvarid (2020), in Anlehnung an Hood (2018).

- **Menüleisten:** Komplette Dropdownmenüs sowie Schnellzugriff auf häufig genutzte Funktionen
- **Perspektivwechsler:** Zum Hin-und-Her-Wechseln zwischen verschiedenen Perspektiven
- **Paket-Explorer-Bereich:** Bereich, in dem die Projekte bzw. Dateien des Benutzers aufgeführt werden
- **Editor-Bereich:** Bereich, in dem der Quellcode editiert wird
- **Verschiedene Bereiche:** Komponenten, wie Konsole, Compilerprobleme in Listenform
- **Aufgabenliste-Bereich:** eine Liste von Aufgaben, die fertiggestellt werden müssen
- **Outline-Bereich:** hierarchische Ansicht einer Quelldatei

Jupyter Notebook

Jupyter Notebook ist eine internetbasierte, interaktive Programmieranwendung, die zur Entwicklung, Dokumentation und Ausführung von Code sowie zur Kommunikation der Ergebnisse eingesetzt wird (Jupyter Team, o. D.). Jupyter Notebook wird hauptsächlich von Datenwissenschaftlern und -wissenschaftlerinnen zur Ausführung von Python-Code verwendet, aber es unterstützt ca. 40 verschiedene Programmiersprachen. Im Allgemeinen kann es zur Datenbereinigung und -transformation, zur numerischen Simulation, zur statistischen Modellierung, zur Datenvisualisierung, zum maschinellen Lernen und vielem mehr verwendet werden. Jupyter Notebook vereint zwei Funktionen; Zum einen enthält

es eine Webanwendung (oder ein browserbasiertes Tool) zur interaktiven Verwaltung von Dokumenten, z. B. beschreibendem Text, Mathematik, Berechnungen und die dazugehörigen grafischen Medien, wie Diagramme oder 3D-Visualisierungen.

Die Webanwendung ermöglicht das Editieren von Texten in einem Browser mit automatischer Syntaxhervorhebung, die Ausführung von Code aus dem Browser, das Darstellen der Berechnungsergebnisse in PNG, SVG, HTML und anderen Rich Media-Formaten, und dabei werden auch mathematische Notationen mit LaTeX nicht vergessen. Und zweitens enthält Jupyter Notebook Notizbuchdokumente, durch die alle in der Webanwendung sichtbaren Inhalte dargestellt werden können. Dazu gehören Inputs und Outputs der Berechnungen, beschreibender Text, mathematische Formeln, Bilder und grafische Darstellungen von Objekten und Quellcode. Diese Dokumente werden als **JSON-Datei** mit der Dateierweiterung .ipynb gespeichert. Die Speicherung im JSON-Format ermöglicht die Versionsverwaltung des Codes. Zudem kann eine .ipynb-Datei, die auf einer öffentlichen URL verfügbar ist, mühelos mit anderen geteilt werden (Jupyter Team, o. D.).

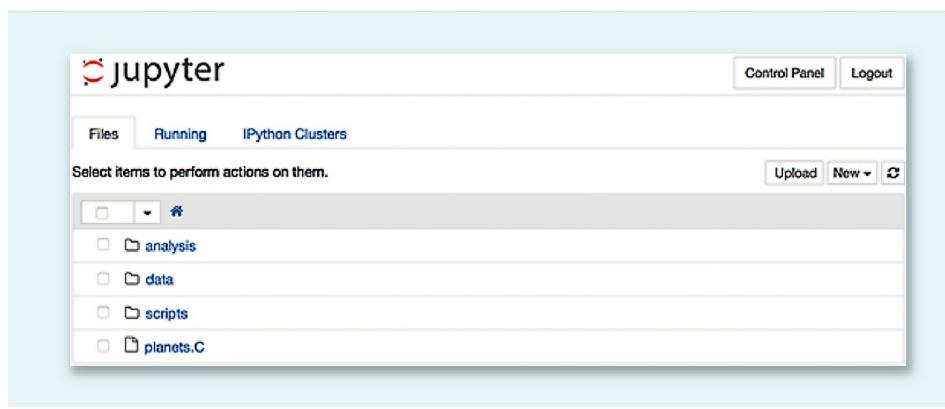
JSON-Dateien

Reine Textdateien, deren Ziel nur der Austausch von Daten zwischen verschiedenen Anwendungen ist.

Einsatz von Jupyter Notebook:

1. Ein Notebook-Server kann von der Befehlszeile mit dem folgenden Befehl gestartet werden (Jupyter Team, o. D.):
`jupyter notebook`
2. Die Landingpage der Jupyter Notebook-Webanwendung ist standardmäßig `http://127.0.0.1:8888` und wird als Dashboard bezeichnet. Sie enthält die im Notizbuch-Verzeichnis enthaltenen Notizbücher.

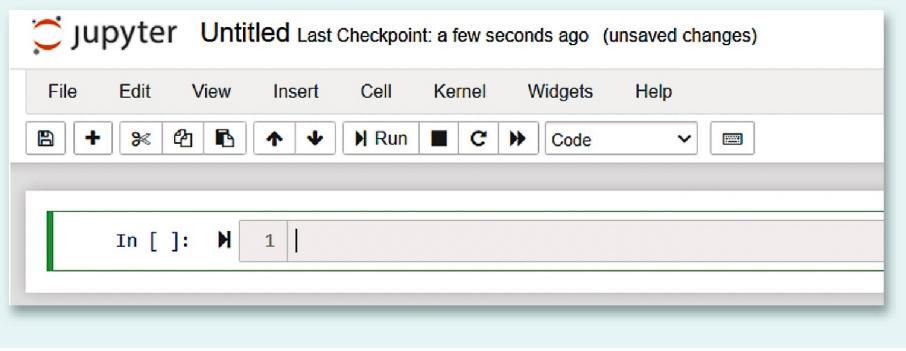
Abbildung 41: Landing Page von Jupyter Notebook



Quelle: Alvarid (2020).

3. Im Dashboard lassen sich unter Neu/Notizbücher neue Notizbücher erstellen. Das neue Notizbuch, in der Form einer .ipynb-Datei, wird unten abgebildet.

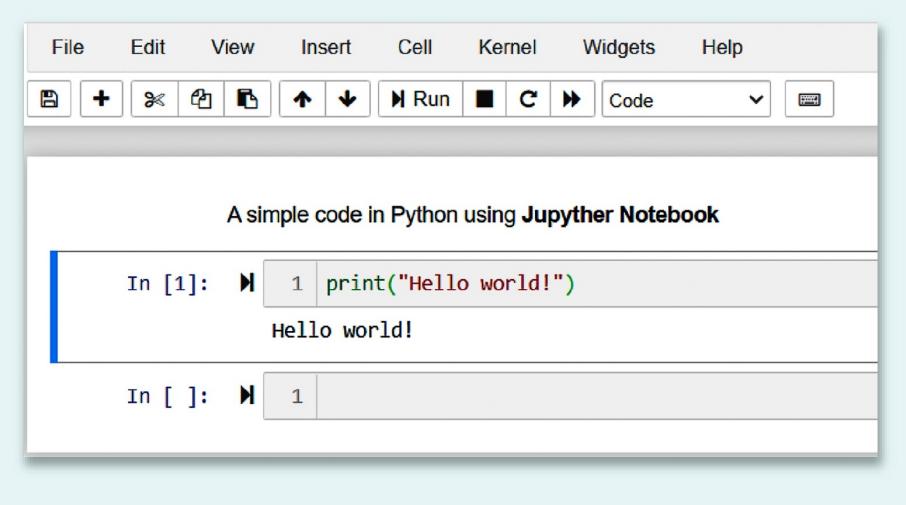
Abbildung 42: Ein Jupyter Notebook (.jpynb Datei)



Quelle: Alvarid (2020).

4. Danach kann ein:e Benutzer:in im Notizbuch codieren.

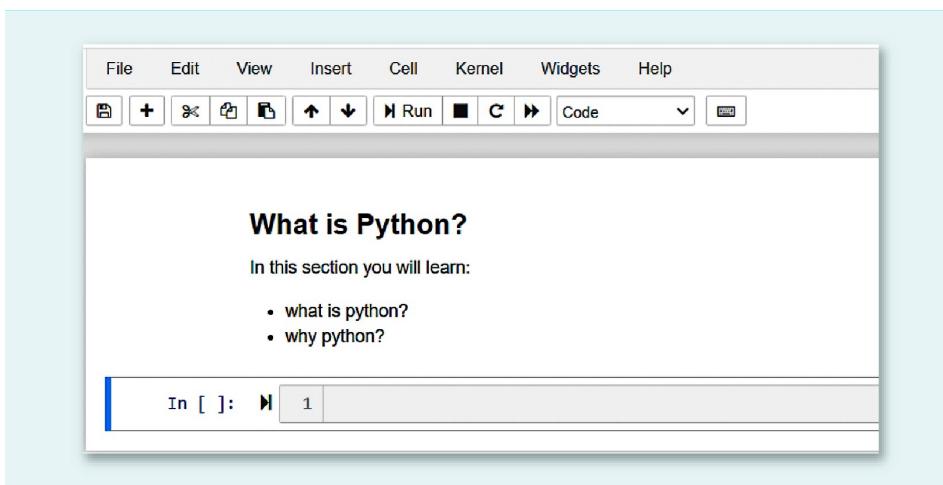
Abbildung 43: Verwendung von Jupyter Notebook zur Ausführung eines einfachen Python-Codes



Quelle: Alvarid (2020).

5. Der Code kann in natürlicher Sprache mit Markdown dokumentiert werden. Wie die folgende Abbildung zeigt, wird auch das Textmarkup (Kursiv, Fett, Aufzählungen usw.) unterstützt.

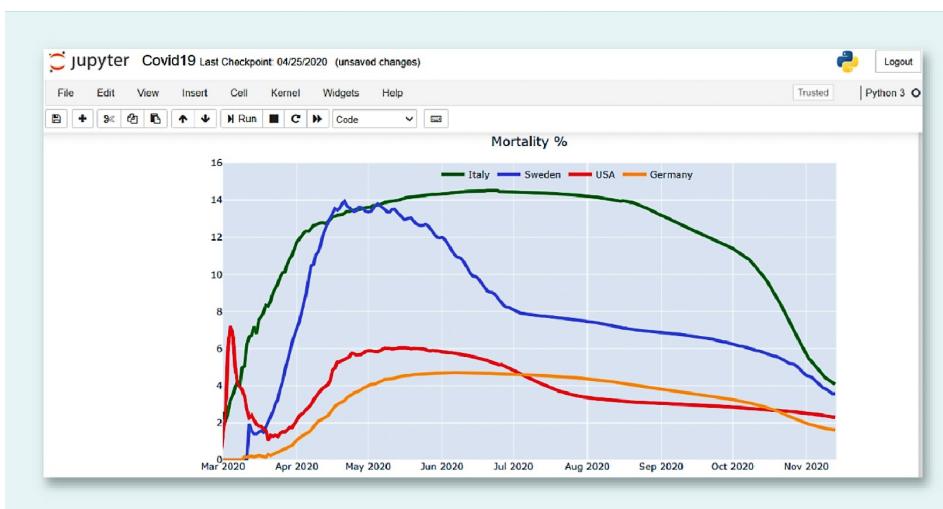
Abbildung 44: Verwendung von Markdown zur Dokumentation von Klartextcode in einem Jupyter Notebook



Quelle: Alvarid (2020).

6. Mit Jupyter Notebook lassen sich auch Liniendiagramme erstellen, wie nachstehend dargestellt.

Abbildung 45: Grafische Darstellung von Daten in Jupyter Notebook



Quelle: "Quelle konnte nicht eindeutig ermittelt werden, bitte redaktionell prüfen"



ZUSAMMENFASSUNG

In dieser Lektion wurden die verschiedenen Ansätze zur Softwareentwicklung behandelt, wobei der Schwerpunkt auf dem Testen lag. Zunächst beschäftigten wir uns mit verschiedenen Testszenarien, wie

Modul-, Integrations-, System-, funktionalen und nicht-funktionalen Tests (nach den Kategorien im Testquadranten von Marick). Anschließend wurde besprochen, wie ein Machine-Learning-System als nicht-deterministisches Modell sowie Daten- und Lernprogramme von ML-Modellen getestet werden können. Es folgte eine kurze Einführung in drei Ansätze, mit denen parallel zur Entwicklung getestet werden kann: Dazu gehörten die testgetriebene Entwicklung (TDD), die verhaltensgetriebene Entwicklung (BDD) und die akzeptanztest-getriebene Entwicklung (ATDD).

Schließlich befassten wir uns mit der Automatisierung von Projektentwicklungspipelines mithilfe von kontinuierlicher Integration, kontinuierlicher Lieferung und kontinuierlichem Testen speziell für Machine-Learning-Pipelines. Für kontinuierliche Integration und Lieferung in einer ML-Pipeline gibt es sechs Schritte: Entwicklung und Experiment, kontinuierliche Integration der Pipeline, kontinuierliche Lieferung der Pipeline, kontinuierliches Training, kontinuierliche Lieferung des Modells und Monitoring. Daraufhin wurden die Prinzipien von Versionsverwaltungssystemen für Softwareprojekte und zwei der gebräuchlichsten Lösungen vorgestellt: Git und GitHub. Und schließlich wurde auf Entwicklertools wie die Befehlszeilenschnittstelle und die integrierte Entwicklungsumgebung (IDE) eingegangen.

LEKTION 4

API

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

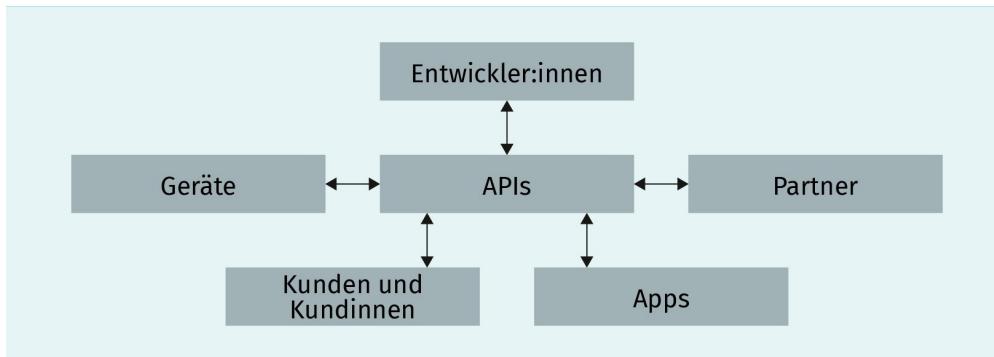
- die verschiedenen Interaktionsarten zwischen Software und Diensten zu verstehen.
- die wichtigsten Prinzipien für das Design und die Erstellung von Programmierschnittstellen zu verstehen.
- zu verstehen, wie sich gutes von schlechtem Schnittstellendesign unterscheidet.
- zu verstehen, wie Bibliotheken in Python mit einem guten Design erstellt werden.

4. API

Einführung

Durch die zunehmende Digitalisierung und Vernetzung haben viele Menschen Zugriff auf intelligente Geräte, die sie für eine immer größere Anzahl von Anwendungen verwenden. Das heißt, dass häufig selbst technisch unerfahrene Benutzer:innen eine komplexe Software verwenden müssen. Damit dies gelingt, muss der Interaktion mit dem Computer ein gutes und leicht verständliches Design zugrunde liegen. Darüber hinaus sollte die Benutzeroberfläche einer Software stabil und zuverlässig sein, sodass die Dienste, die wir in Anspruch nehmen, reibungslos funktionieren und nicht plötzlich abbrechen. Das Einzelhandelsgeschäft von Amazon soll als ein Beispiel dienen. E-Commerce-Plattformen, wie die von Amazon, sind weit verbreitet. Wahrscheinlich haben Sie selbst dort schon einmal nach Produkten gesucht, etwas ausgewählt und in den Einkaufswagen gelegt. Die Website, auf der wir uns intuitiv bewegen, stellt lediglich die Benutzeroberfläche eines hochkomplexen Software- und Hardware-Backends dar, das weltweit von Zehntausenden Maschinen und unzähligen Ingenieur:innen, die diese Plattform in den letzten beiden Jahrzehnten aufgebaut haben, betrieben wird. Da wir diese Dienste und Anwendungen heute tagtäglich nutzen, wird schnell vergessen, wie groß die Diskrepanz zwischen dem, was wir sehen, und dem Rest sein kann. Die Interaktion der Benutzer:innen mit der Anwendung kann als die Spitze des Eisbergs einer komplexen Software aufgefasst werden. Manchmal kann bereits die Benutzeroberfläche überfordern oder schlichtweg kein gutes Design haben. Auf einer Benutzeroberfläche ist jedoch nicht nur das wichtig, was wir sehen, sondern auch das, was eben nicht gezeigt werden soll. Softwareingenieur:innen und Datenwissenschaftler:innen arbeiten täglich am Computer. Und obwohl ihre Interaktion mit den Systemen anders aussieht als bei Durchschnittsnutzer:innen, so müssen sie sich dennoch auf die Verträge mit den Programmen und Tools verlassen können. Diese Verträge sind die so genannten Anwendungsprogrammierschnittstellen, kurz APIs. Auch die meisten Hobbyprogrammierer:innen kennen heute den Ausdruck API. Um auf Amazon zurückzukommen, so lässt sich sagen, dass die Veröffentlichung des Technologie-Stacks über APIs in den Amazon Web Services (AWS) inzwischen ein genauso großes Geschäft wie der Einzelhandel von Amazon ist. Nicht jeder Klick auf einer Website ist ein API-Aufruf, aber jede Geschäftstransaktion eines:einer Benutzer:in entspricht einem solchen. Es lässt sich durchaus eine API-zentrierte Ansicht der modernen, digitalisierten Geschäftstransaktionen vertreten, bei der nicht nur API-Entwickler:innen, sondern auch Anwendungen und Geräte, Kund:innen und strategische Geschäftspartner:innen über APIs interagieren.

Abbildung 46: API-zentrische Ansicht



Quelle: Pumperla (2020).

Diese Lektion erteilt einen Überblick der geläufigsten API-Beispiele und somit der zahlreichen Ansätze zur produktiven Interaktion mit Computern. Dabei konzentrieren wir uns auf eine bestimmte Benutzer:innengruppe, und zwar auf Ingenieur:innen, Wissenschaftler:innen sowie die Teilnehmenden dieses Kurses. Neben dem Verständnis für ein Computerprogramm sollten erfahrene Fachkräfte lernen, sich kritisch mit Programmierschnittstellen auseinanderzusetzen. Denn am Ende des Programmierens steht eine Anwendungsschnittstelle, die von Benutzer:innen verwendet wird. Daher ist es lohnenswert, sich die Fähigkeiten, die für gute Schnittstellen notwendig sind, zu erarbeiten. Die Entwicklung und die Anwendung von APIs gehören eng zusammen. Daher entwickeln wir einige API-Beispiele in Python, um zu zeigen, wie gutes Schnittstellendesign in der Praxis aussieht. Gleichzeitig schauen wir uns Gegenbeispiele an, von denen man lernen kann. Um es mit Martin Fowlers Worten zu sagen: „Jeder Idiot kann Code schreiben, der von Computern verstanden wird. Gute Programmierer:innen schreiben Code, den Menschen verstehen können“ (Fowler & Beck, 1999, S. 15, vom Autor übersetzt).

4.1 Interaktion mit Software und Diensten

Moderne Computer sind äußerst komplex. Selbst Expert:innen können bei der Interaktion mit dem Computer nur deshalb produktiv sein, weil es viele Abstraktionsebenen gibt, die geschickt verstecken, was im Hintergrund passiert. Wer eine Analyse in Python durchführt, sollte sich auf den speziellen Anwendungsfall konzentrieren und nicht darauf, wie der Code in Nullen und Einsen übersetzt wird oder wie der CPU vom Betriebssystem zur Analyseausführung eingesetzt wird. Letztendlich muss man sich darauf verlassen, dass der Code als ein kleiner Teil aller Programmierfunktionen, genau das tut, was er soll. Daher verwenden wir ständig Anwendungsprogrammierschnittstellen, um die Komplexität zu verringern und um produktiv sein zu können. Wie bereits erwähnt, ist eine API eine Schnittstelle, die den Austausch zwischen einer oder mehreren Parteien und einer Softwarekomponente ermöglicht. Genauer gesagt muss eine API ...

- ... die möglichen Anfragen oder Anrufe definieren,

- das als Input erwartete Datenformat und die erstellten Formate klar darlegen und
- kommunizieren, welche Konventionen befolgt werden und was das gewünschte Verhalten jeder Anfrage ist.

Als erstes konkretes Beispiel einer Klasse von API-Aufrufen betrachten wir die Funktion von Python zum Import von Bibliotheksprogrammen. Diese dreht sich um ein einziges Schlüsselwort, nämlich `import`. Mit `import` können drei essenzielle Anrufe ausgeführt werden: Import von Python-Modulen, Import von Funktionalität aus Modulen und Alias-ing. In dieser Lektion kommt Python 3.7 bzw. spätere Versionen zum Einsatz. Hier ist ein Beispiel:

Code

```
import time # import the time module
from time import ctime # import the "current time"
import time as t # refer to time module as t
```

Zur Verwendung von `import` muss dem stets ein gültiger Python-Paketname folgen. Das kann entweder ein integriertes Paket oder ein Paket von einem Drittanbieter sein, das in der Python-Umgebung installiert ist. Eine `import`-Anweisung gibt bei Erfolg nichts zurück. Man spricht davon, dass sie lautlos ist. Wenn die zu importierende Bibliothek nicht existiert, wird eine Fehlermeldung ausgegeben. Bei Eingabe von `import testfailure` in der Python-Sitzung erscheint beispielsweise folgende Fehlermeldung:

Code

```
ModuleNotFoundError: No module named 'testfailure'
```

Wie zu erwarten, soll `import` einen Import von Python-Funktionalität durchführen und sie dann dem:der Benutzer:in in der laufenden Sitzung oder während der Skriptausführung zur Verfügung stellen. Die obigen Importanweisungen würden es dem:der Benutzer:in beispielsweise ermöglichen, die aktuelle Systemzeit mit `ctime()` abzufragen. Diese wird dem:der Benutzer:in dann in der Form eines Strings in Python zurückgegeben. Bevor wir tiefer in APIs und in für die Arbeit von Entwickler:innen und Datenwissenschaftler:innen relevante Typenbeispiele von APIs einsteigen, werfen wir einen Blick auf die Entwicklung des API-Begriffs und darauf, was er heute bedeutet.

Geschichte und Entwicklung von APIs

Der Terminus API hat seit seiner Einführung eine interessante Entwicklung durchlaufen. Aus heutiger Sicht haben Programmierer:innen schon mit den ersten Programmen APIs entwickelt. Ursprünglich wurde die Benennung API nur für Anwendungsschnittstellen zu Benutzer:innen verwendet, allerdings wurde die Bedeutung bald erweitert. Die Bezeichnung selbst wurde erstmals in den 1960er-Jahren veröffentlicht (Cotton & Greatorex, 1968), jedoch dauerte es noch etwas länger, bis der Begriff populär wurde. Cotton und Greatorex (1968) schlugen eine gleichbleibende Anwendungsschnittstelle in Fortran vor, die die Verwendung von grafischen Anwendungen, mit denen sie sich befassten, erleichtern sollte, indem diese auf Hardware aller Art laufen sollte. Zur Verteilung von Software auf verschiedenen Maschinen muss sie Hardware-unabhängig sein. Das ist für moderne

Anwendungsdesigner:innen offensichtlich, aber in den Anfangstagen der Programmierung stellte dies kein leichtes Ziel dar. Im Jahr 1974 dehnte sich der Begriff API auf Datenbankverwaltungssysteme aus, aber die Anwendungsschnittstelle war streng von anderen Interaktionsarten mit der Datenbank, z. B. der Suche, getrennt (Date, 2019). Dies stellte sich als zu strikt heraus, und folgende Generationen erkannten, dass sich durch die Erweiterung des Begriffs alle möglichen Interaktionen vereinen ließen.

Und so wurde die Entwicklung von differenzierten, integrierten Schnittstellen in allen Bereichen der Informatik zur Norm, nicht nur im Design von Datenbanken (Berg et al., 1981). In den 1990er-Jahren war es allgemein anerkannt, den API-Begriff im weiteren Sinne zu verwenden, um alle möglichen Arten von Programmierung abzudecken. Die damalige Definition beschrieb APIs als „eine Reihe von Diensten, die Programmierern zu bestimmten Aufgaben zur Verfügung stehen“ (Malamud, 1990, S. 294). Mit der Einführung des Internets wurden Web-APIs zu den erweiterten „netzwerkbasierten Anwendungsprogrammierschnittstellen“ (Fielding, 2000). Web-APIs sind heute so weit verbreitet, dass „API“, was früher eigentlich einen engeren Begriff abdeckte, als Abkürzung für Web-API und für die Kommunikationsprotokolle verwendet wird. Und tatsächlich wird API häufig für die noch engere Bedeutung von JSON- oder XML-basierten Web-APIs verwendet. In dieser Lektion wählen wir einen holistischeren Ansatz der Anwendungsprogrammierschnittstellen, von denen Web-APIs nur ein Teil sind. Die schnelle Einführung des Internets und die zunehmende Vernetzbarkeit von Benutzer:innen auf der ganzen Welt führte zu einer Explosion von Web-APIs, die sicherlich weiter andauert. Ihr Anfang war jedoch einfach: Zu den Pionieren im kommerziellen Einsatz von Web-APIs gehörten einige Größen der Dotcomwelle wie Salesforce, Ebay und Amazon. Im Jahre 2002 rief Amazon beispielsweise die Amazon Web Services (AWS) ins Leben. Mit diesem Service können Entwickler:innen Inhalte von Amazon in ihre eigenen Websites einbetten. AWS hat sich seitdem rasant weiterentwickelt, und die Cloud-Angebote von Amazon stehen beinahe auf dem Niveau des Vertriebsgeschäfts von Amazon. Mit der nächsten Welle der Web-APIs, die ca. im Jahr 2004 begann, kam die Integration der sozialen Medien durch Tech-Giganten wie Facebook, Twitter oder Flickr. Die Entwickler-API von Twitter wurde im Jahr 2006 veröffentlicht, um Ingenieur:innen Zugriff auf alle möglichen Daten der Plattform zu erteilen. Im selben Jahr wurde die API von Facebook gestartet, mit der Entwickler:innen Zugang zu Informationen von Facebook-Freund:innen, Bildern, Beiträgen usw. erhielten. Dies machte die Plattform für Werbefirmen attraktiv und verhalf Facebook dazu, seinen globalen Einfluss zu erweitern. In einem weiteren Schritt begannen Unternehmen, ihre Infrastruktur und Dienste in die Cloud zu verlagern. Amazon übernahm dabei eine Pionierrolle, als es im Jahr 2006 eine Reihe neuer Dienste, angefangen mit Amazon Simple Storage Service (S3) und Amazon Elastic Compute Cloud (EC2), auf den Markt brachte. Beide Dienste tragen heute noch zum Erfolg von AWS bei. Während Amazon S3 Cloud-Speicher mit einfacher nutzungsbasierter Bezahlung erschlossen hat, ermöglichte Amazon EC2 Entwickler:innen, Zugriff auf Computerressourcen in der Cloud zu erhalten, ihre eigenen Maschinen aufzurüsten und ihre Dienste dort zu hosten. Die Tatsache, dass der Erfolg von AWS zum Teil der API-fokussierten Kultur von Amazon geschuldet ist, ist bemerkenswert (API Evangelist, 2012). Bis heute müssen Amazontteams ihre Funktionen anderen Teams als Dienste zur Verfügung stellen. Obwohl dies kurzfristig einiges gekostet haben dürfte, denn APIs zu bauen ist nicht billig, so wäre die geniale Entscheidung, die Firmeninfrastruktur der Öffentlichkeit als Service anzubieten, ohne sie nicht möglich gewesen. Mit dem Release des iPhones wurde das Internet und damit auch die nächste Generation an APIs mobil.

Zuvor soll noch auf eine weitere bemerkenswerte Veränderung in der Tech-Landschaft eingegangen werden, die sich auf APIs ausgewirkt hat. Mit dem Aufkommen des „Internets der Dinge“ sind immer mehr Geräte mit dem Internet verbunden und kommen daher auch für APIs in Frage. Dazu gehört Alexa von Amazon (2014), die den API-Zugriff für alle relevanten Funktionen, wie die Sprachsteuerung, die Weckfunktion oder den Wetterdienst, ermöglicht. Fitbit (2017) stellt API-unterstützte Wearables zum Messen von Fitnessdaten her. Im historischen Kontext deutet nichts mehr auf die Wichtigkeit von APIs und das kommerzielle Interesse hin, das sie in dieser modernen Zeit darstellen, als der berüchtigte Fall der Tech-Giganten Oracle und Google, die 2017 einen Streit über die Urheberrechte für 37 Java-APIs anstrengten (Tsidulko, 2020). Abschließend lässt sich feststellen, dass APIs und ihre Benutzung in der heutigen Zeit allgegenwärtig sind und sie in vielen für uns relevanten Bereichen eingesetzt werden:

- Hinter fast allen Webanwendungen befinden sich APIs.
- Sie sind der Antrieb von mobilen Anwendungen und Desktopanwendungen.
- APIs liegen der Netzwerkkommunikation zwischen Geräten und Anwendungen über das Internet zugrunde.
- Sie sind die Eckpfeiler des „Internets der Dinge“ und verbinden intelligente Geräte, wie Autos, Kühlschränke oder Staubsauger.

Spezifikationen und Verträge

Für jede Software gibt es eine:n Anbieter:in, also ein Unternehmen oder eine Gruppe von Personen, die für den Entwurf, die Entwicklung und Pflege verantwortlich sind. Zudem gibt es Verbraucher:innen, die die Software für ihre Bedürfnisse nutzen. Verbraucher:innen können entweder Personen oder andere Maschinen sein, die die Software nutzen. Eine API kann, ganz gleich welche tatsächliche Rolle sie spielt, als Spezifikation oder Vertrag zwischen Anbieter:in und Verbraucher:in betrachtet werden. Wie auch bei anderen Verträgen so müssen sich die Parteien zunächst auf den Vertragsinhalt einigen und dies festhalten. Im Software-Geschäft kann der:die Anbieter:in bis zu einem gewissen Grad vorgeben, wie eine Schnittstelle aussieht, er sollte jedoch aufpassen, die Kund:innen nicht zu verlieren, wenn er:sie z. B. an schlechtem Design festhält. Wenn sich die Parteien auf eine API einigen, die als Vertrag betrachtet wird, müssen sie sich stets daran halten. Firmen, die „Software as a Service“ (SaaS) anbieten, treffen meist Service-Level-Vereinbarungen (SLAs), in denen die Beständigkeit der APIs aufgeführt ist. So kann eine Firma die Struktur von API-Aufrufen nicht von heute auf morgen ändern. Das zieht bestimmte Probleme in Bezug auf die Wartung und Erweiterbarkeit der APIs nach sich. Ähnlich wie Softwarepakete oder Betriebssysteme hat jede API eine Version. Die Entwicklerplattform GitHub hat eine Entwickler-API, deren URL folgendermaßen lautet

Code

<https://developer.github.com/v3/>

In der URL befindet sich das Suffix v3: Es steht für die derzeitige, stabile Version der GitHub-API, historisch bezeichnet es jedoch die dritte Hauptversion der API. Die ersten beiden sind veraltet und wurden vor einigen Jahren entfernt. Die Versionierung von APIs hat viele Vorteile:

- In einer Version garantiert der:die Anbieter:in, dass jeder einzelne API-Aufruf eine stabile Schnittstelle hat. Die Benutzer:innen der API, die nachgeschaltete Anwendungen entwickeln, können sich darauf verlassen, dass sie nicht plötzlich und ohne Vorwarnung ausfällt.
- Einer API-Version lassen sich neue Funktionen hinzufügen, wenn sie nicht an alte Anforderungen gebunden ist. In Bezug auf eine Software würde man von Nebenversionen oder Software-Patches sprechen. Obwohl die Fehlerbehebung Teil der Pflege zur Aufrechterhaltung von APIs ist, bietet sich diese Art der Erweiterbarkeit in einem Release oft an.
- Neue API-Versionen kommen erst bei grundlegenden Änderungen zum Tragen. Wenn Code, der eine API in einer alten Version nutzt, mit einer neuen Version nicht funktioniert, spricht man von Breaking Change. API-Entwickler:innen sollten versuchen, Schnittstellen gleich beim ersten Mal richtig zu programmieren, aber das ist natürlich nicht immer möglich. Designentscheidungen für eine Benutzerschnittstelle sollten dennoch gut durchdacht sein, denn sie lassen sich nicht so leicht ändern wie interne Prozesse. Die Versionsverwaltung von APIs ermöglicht es Anbieter:innen, mehrere Versionen der API gleichzeitig zu veröffentlichen und Verbraucher:innen genügend Vorbereitungszeit zu geben, so dass sie planen können, wann und wie sie auf die neueste API-Version umsteigen. Vorbereitungszeit ist für Benutzer:innen äußerst wichtig und plötzliche Änderungen sollten vermieden werden.
- Die Benutzer:innen über den Ablauf älterer API-Versionen zu informieren und diese gleichzeitig zu warten, stellt eine zusätzliche Belastung für die Anbieter:innen dar, den Verbraucher:innen kommt es aber zugute.
- API-Versionen können **veraltet** sein, wenn sie Programmfehler enthalten, ineffizient sind oder schlechte Codierpraktiken fördern. Individuelle Anrufe können ganz aus einer neuen Version entfernt werden, wenn sie nicht mehr gebraucht werden.

Dokumentation

Ein Thema, das im Rahmen von API-Verträgen besondere Aufmerksamkeit verdient, ist die Dokumentation. Anwendungsentwickler:innen betrachten die Dokumentation oft als notwendiges Übel und widmen sich lieber ganz dem Codieren. Diese ist jedoch zur Einhaltung eines gemeinsamen Verständnisses einer API unabdingbar. Ganz gleich, wie einfach man sich eine Programmierschnittstelle vorstellt, sie muss schlussendlich von einer Person bedient werden können, selbst wenn es nur darum geht, einer anderen Maschine Anweisungen zu erteilen. Und dabei besteht stets das Risiko, dass Missverständnisse entstehen. Manche gehen so weit zu sagen, dass die API die Dokumentation darstellt und der Code deren Implementierung ist, was aus Sicht der Softwareplanung durchaus einleuchtet. Bevor das Manifest für Agile Softwareentwicklung eingeführt wurde, war es in der Informatik üblich, explizite und äußerst detaillierte Spezifikationen (kurz Specs) zu schreiben, an die man sich halten musste, bevor mit der Codierung oder der Produktion begonnen werden konnte. Die Spezifikation – für eine API oder ein anderes Produkt – kann als Ergebnis einer möglicherweise langen Verhandlungsphase angesehen werden. Im Projektmanagement wird dieser Prozess oft als Anforderungsentwicklung beschrieben. Aus praktischer Sicht ist die Dokumentation nicht zu unterschätzen. Wenn man Murphys Gesetz auf APIs anwenden würde, so ließe sich sagen, dass jeglicher API-Aufruf, der missverstanden werden kann, missbraucht wird. Daher sollte bei der Erstellung und Dokumentation von Programmierschnittstellen so eindeutig und präzise wie möglich vorgegangen werden.

Veraltete Anwendungen (Deprecation)

Eine API kann einige Aufrufe als veraltet deklarieren, d. h. der Teil wurde entweder geändert oder entfernt. Warnungen vor veralteten Anwendungen ermöglichen es Entwickler:innen, zu Teilen der API überzuwechseln, die langfristig unterstützt werden.

Sicherheit, Governance und Release-Richtlinien

APIs ermöglichen es Unternehmen, Technologien, die außerhalb ihres Betriebs entstanden sind, zu nutzen, wodurch Abhängigkeiten entstehen. Gleichzeitig muss sich ein Unternehmen, das den Zugriff zu einer API bereitstellt, plötzlich mit externen Abhängigkeiten auseinandersetzen. Beides bringt nicht nur administrative Probleme mit sich, sondern birgt auch potenzielle Sicherheitsrisiken. Wenn sich ein Unternehmen auf die Verfügbarkeit einer API verlässt und sich diese jedoch verlangsamt, wird das Geschäft aufgehalten. Daher wird die Leistungskontrolle der externen API zu einem integralen Bestandteil der internen Geschäftsprozesse. Wenn eine Integration plötzlich nicht mehr funktioniert, muss umgehend herausgefunden werden, ob der Fehler auf eine interne Änderung oder eine des externen Anbieters zurückzuführen ist. Der interne Entwicklungszyklus muss deshalb den externen API-Entwicklungszyklus, von dem er abhängt, berücksichtigen. Es kommt nicht selten vor, dass große Firmen eigene API-Manager:innen engagieren, die alle (externen) API-bezogenen Angelegenheiten innerhalb eines Unternehmens planen und verwalten, z. B. die Einhaltung betrieblicher oder nationaler Normen durch die integrierten Tools. Wenn dagegen eine API öffentlich zugänglich gemacht wird, kann es vorkommen, dass Produktionsdaten des Unternehmens zugreifbar und dadurch auch angreifbar werden. So müssen beispielsweise Datenanbieter im Gesundheitswesen ihre Daten in den meisten Ländern zwingend in anonymisierter Form verwalten. Die Sicherheitsrisiken von Unternehmen mit vertraulichen Daten können gravierend sein. Daher lohnt es sich, in ausreichende Sicherheitsstandards zu investieren und angemessene Zugriffsrechte für die APIs festzulegen. Im Großen und Ganzen gibt es drei wesentliche Richtlinien, unter denen APIs freigegeben werden können (Boyd, 2014):

- **Privat:** Die API wird nur firmenintern verwendet.
- **Partner:innen:** Die API wird nur einer bestimmten Benutzergruppe, d. h. Geschäftspartner:innen, zur Verfügung gestellt.
- **Öffentlich:** Die API ist öffentlich verfügbar. In typisierten Programmiersprachen wird das Stichwort „**öffentlicht**“ oft für Klassen oder Funktionen verwendet. Das heißt, dass dieser Teil der entsprechenden Bibliothek allen Programmierer:innen, die mit dieser Bibliothek arbeiten, zugänglich ist. Diese Art „**öffentlicher**“ Funktionalität kann dennoch Teil einer privaten, innerhalb einer Firma verfügbaren API sein. „**Öffentlich**“ heißt nicht „umsonst“ oder dass sie für alle verfügbar ist. Es bedeutet, dass diejenigen zur Verwendung berechtigt sind, die über einen Zugriff verfügen und für diesen bezahlen (Fowler, 2002).

Veröffentlichte im Vergleich zu öffentlichen Schnittstellen
Nur wenn eine API veröffentlicht wird, also der Öffentlichkeit über das Internet Zugriff erlaubt wird, wird sie im hier beschriebenen Sinn mit allen Vor- und Nachteilen „**öffentlich**“.

Einer der bereits behandelten Vorteile von APIs besteht darin, dass die Schnittstelle innerhalb eines API-Releases stabil ist. API-Designer markieren ihre Aufrufe oft als instabil, um sich etwas mehr Flexibilität zu verschaffen. In der Guava Bibliothek von Google Java werden instabile Komponenten mit @Beta annotiert, womit eine niedrige Zuverlässigkeit der Komponente angezeigt wird, da sie sich ändern oder ganz verschwinden könnte.

API-Arten

Nachdem wir unterschiedliche Beispiele, die Geschichte von APIs und einige der wichtigsten Aspekte, die APIs ausmachen, besprochen haben, sollen verschiedene Ebenen und Arten von APIs beleuchtet werden. Die Betrachtung erfolgt gewissermaßen „von innen

nach außen“: So fangen wir mit den Komponenten auf niedrigeren Ebenen eines Computers an, um dann auf die höheren Ebenen des Anwendungsdesigns einzugehen. Diese zwiebelartige Schichtung der API-Abstraktionen ermöglichen die effiziente Arbeit an einem Computer. Das vorliegende Studienskript, das Sie gerade lesen, wurde in einem Textverarbeitungssystem geschrieben, ohne dass dabei auch nur eine Sekunde an die zugrundeliegende Hardware gedacht wurde.

Betriebssysteme

Einfach ausgedrückt, ist ein Betriebssystem (kurz: OS für Operating System) eine Software, die Hardwarekomponenten und Softwareressourcen verwaltet und steuert. Zudem stellt es herkömmliche Dienste der Systemebene, die die Benutzer:innen brauchen, zur Verfügung. Die **POSIX**-Norm ging in den 1980er-Jahren aus der Arbeit von Richard Stallman hervor und vereinte verschiedene UNIX-Systeme unter einem Dach. Diese Norm ist die Basis des Betriebssystems Linux (Stallman, 2011). Darin werden eine Reihe von APIs festgelegt, die in POSIX-konformen Betriebssystemen implementiert sein müssen, sodass komplizierte, auf diesen APIs beruhende Programme auf anderen POSIX-Systemen ausgeführt werden können. Linux-Benutzer:innen treffen wahrscheinlich täglich auf POSIX und zwar im Zusammenhang mit den Datei- und Verzeichnisformaten, Pipes und Spezifikationen für Input und Output (I/O-Portschnittstelle und -steuerung). Betriebssysteme sind komplexe Software und man könnte noch viel über sie schreiben, vor allem in Bezug auf APIs, dies würde jedoch den Umfang dieses Studienskripts überschreiten. Folgende Beispiele sind ebenfalls bemerkenswerte APIs, die eng mit der Betriebssystemebene zusammenarbeiten und für datenintensive Wissenschaften wichtig sind:

- OpenGL (Open Graphics Library), eine sprach- und plattformübergreifende API für hardwarebeschleunigtes Rendering von 2D- und 3D-Vektorgrafiken, die häufig bei der Entwicklung von Videospielen, Computer-Aided Design (CAD) und Anwendungen der virtuellen Realität im Einsatz sind. Die im Jahre 1992 eingeführte Norm wird heute durch das Non-Profit Konsortium der Khronos Group verwaltet.
- OpenCL (Open Computing Language), ein Framework und eine API für das Schreiben von Anwendungen unabhängig von besonderen Plattformen und für den Einsatz verschiedener Hardwarekomponenten wie Zentralprozessoren (CPUs), Graphical Processing Units (GPUs) und Field Programmable Gate Arrays (FPGAs). OpenCL stellt C/C++-basierte Programmiersprachen sowie API-Aufrufe zur Parallelverarbeitung auf Geräten zur Verfügung. Genau wie OpenGL wird auch OpenCL von der Khronos Group gepflegt. Alle großen Hardwarehersteller außer Apple, das die Unterstützung von OpenCL benötigt hat und andere Normen unterstützt, bieten OpenCL-konforme Hardware an.
- CUDA (Compute Unified Device Architecture), eine von Nvidia für die eigene Hardware entwickelte API für Paralleldatenverarbeitung auf GPUs. CUDA funktioniert zum Beispiel mit C/C++ oder Fortran. Die Programmierung ist nicht trivial, aber sie hat im Vergleich zu OpenGL Vorteile. Effiziente und schnelle Implementierungen von Matrixoperationen, wie Matrizenmultiplikation und ähnlichen höherdimensionalen Operationen, sind unter anderem für den Erfolg auf dem Gebiet des Deep Learning verantwortlich.

POSIX

Das Portable Operating System Interface (POSIX) ist eine Reihe von Normen, die von der IEEE Computer Society festgelegt wurden. Durch POSIX wird eine API auf Betriebssystemebene mit Pipelines, Prozessen, Shells, Dienstprogrammen und vielen mehr definiert.

Befehlszeilenschnittstelle

Shell

Ein Befehlszeileninterpretierter, der Zugriff auf die API des Betriebssystems hat.

Obwohl Shell sich auch auf die grafische Benutzeroberfläche beziehen kann, wird der Ausdruck meistens für die Befehlszeilenschnittstelle (CLI) verwendet. Er kommt daher, dass Shells die äußersten Schichten des Betriebssystems sind.

Regelmäßige Nutzer:innen von UNIX-Systemen, wie Linux, MacOS, aber auch Poweruser von Windows führen einen großen Teil ihrer Programme in einer Befehlszeilen-Shell bzw. einer **Shell** aus. Tools, die in der Shell ausgeführt werden, werden als Befehlszeilenschnittstelle (CLIs) bezeichnet und sind eine Sonderart der Betriebssystemressourcen, die APIs nutzen. CLIs sind einfach, jedoch mächtige Systeme auf Textbasis, die die Anweisungen Zeile für Zeile ausführen. Unter den vielen CLI-Tools, die auf den verschiedenen Betriebssystemen zur Verfügung stehen, müssen zwei für Python Entwickler:innen besonders wichtige Tools genannt werden, nämlich `python` selbst und der Paket-Manager `pip`. Eine interaktive Sitzung kann über die simple Eingabe von `python` in der Shell eines Systems gestartet werden. Dieser Befehl kann auch zum Aufruf von Python-Programmen oder zur Ausführung von als Python-Code interpretierbaren Strings direkt aus der Shell verwendet werden. Durch folgenden Befehl wird zum Beispiel die Zahl 42 als Output in die Shell mit Hilfe der Python-CLI ausgedruckt:

Code

```
python -c "print(42)"
```

Durch die Flag `-c` kann jeder String, der von einem Python-Interpreter gelesen werden kann, weitergegeben werden. Ein weiteres Verwendungsbeispiel von `python` als CLI-API ist die Suche nach der auf dem System installierten Version durch die Eingabe von `python -V`. Um standardmäßig Pakete für Python zu installieren, kann der Python Paketindex (PyPI) über das CLI-Tool `pip` (obwohl es auch andere Tools gibt, wie das altbekannte `easy_install`, das heute populäre Poetry oder conda, das mit allen Extras ausgestattet ist) verwendet werden.

Um die beliebte Bibliothek `pandas` zur Datenanalyse zu installieren, kann folgender Befehl verwendet werden:

Code

```
pip install pandas
```

Zur Aktualisierung von `pip`, das nur ein Python-Paket auf PyPI ist, kann dieser Befehl eingegeben werden:

Code

```
pip install --upgrade pip
```

Es sollte erwähnt werden, dass Entwickler:innen im Kontext von Web-APIs oft davon sprechen, eine API und eine CLI für einen bestimmten Dienst zu haben. „API“ bezieht sich in diesem Fall meistens auf eine Webschnittstelle oder den Zugriff über eine Programmiersprache, während „CLI“ der Teil der API ist, auf den über die Shell zugegriffen wird.

Programmiersprachen

Es wurde dargelegt, wie Python als CLI genutzt werden kann, um Befehle in der Shell auszuführen. Aber natürlich ist Python eine vollständige Programmiersprache, die für ihr elegantes Design und die hohe Nutzerfreundlichkeit gelobt werden sollte. Hier gehen wir nicht weiter auf Einzelheiten von Python ein, wollen aber anmerken, dass alle Datenarten, Variablen, integrierten Funktionen und Klassen dieser Sprache tatsächlich selbst APIs zur Interaktion mit Benutzer:innen sind. Vor allem Anfänger:innen ist es weniger bekannt, dass die Python-API viele verschiedene Implementierungen hat. Wenn ein:e Entwickler:in sagt, dass er:sie in Python programmiert, bezieht er sich vermutlich auf CPython, eine Open-Source-Implementierung der Python-API, die auf der Programmiersprache C basiert, weit verbreitet ist und kostenlos zur Verfügung steht. Es gibt auch Alternativen wie Jython (in Java), PyPy (in RPython) und IronPython (in C#). Bei der Arbeit mit einer beliebigen Programmiersprache gerät es leicht in Vergessenheit, dass es sich dabei um die Implementierung einer Designspezifikation handelt, die in einer anderen Programmiersprache geschrieben wurde. Python ist eine ausdrucksstarke Sprache, aber letztlich handelt es sich dabei lediglich um eine API für Code auf niedriger Ebene. Dieses Wissen ist nützlich, da die Arbeit mit CPython es Ihnen ermöglicht, in C geschriebene Erweiterungen von Python zu schreiben, und das Ergebnis könnte sogar schneller als gewöhnliches Python sein. Beliebte Bibliotheksprogramme, wie NumPy, ein Python-Paket zur effizienten Manipulation n-dimensionaler Arrays, werden im Allgemeinen als eine C-Erweiterung von Python geschrieben.

Programmbibliotheken und -frameworks

Bewegen wir uns aus dem Zwiebelinneren noch um eine weitere Schicht nach außen. Die in der Praxis verwendeten Bibliotheksprogramme und Frameworks – z. B. Bibliotheken für die Datenanalyse und -visualisierung – sind selbst Erweiterungen der Programmiersprache, in der sie geschrieben sind. NumPy wurde bereits genannt: Es ist eine API für n-dimensionale, in Python geschriebene Arrays. Man sollte sich also keinesfalls darüber den Kopf zerbrechen, wie NumPy funktioniert. Man muss lediglich konzeptionell verstehen, was diese Schnittstelle ist, die NumPy zur Verfügung stellt. In anderen Worten ist die Schnittstelle von den Implementierungsdetails getrennt. Dieses wichtige Grundprinzip aller APIs untersuchen wir im nächsten Abschnitt. Zuvor sei noch ein einfaches Beispiel genannt: Wir möchten eine 3x3-Matrix in Python erstellen. Diese sollte mit sich selbst addiert und das Ergebnis ausgedruckt werden. Mithilfe von NumPy kann das folgendermaßen aussehen:

Code

```
import numpy as np
x = np.ones((3, 3))
print(x + x)
```

Wir erhalten das erwartete Ergebnis

Code

```
[[2. 2. 2.]
 [2. 2. 2.]
 [2. 2. 2.]]
```

Dazu muss man nur wissen, dass der Funktionsaufruf der NumPy-API, die Matrizen mit der Zahl 1 in allen Einträgen und in der bestimmten Form (hier 3x3) erstellt, ones lautet und dass Matrizen durch den Operator „+“, der in Python überschrieben werden kann, addiert werden können. Unwichtig ist es zu wissen, wie Matrizen in NumPy gespeichert werden oder wie Matrizenadditionen oder andere Operationen ausgeführt werden. Dadurch wird die kognitive Belastung gemindert, und Sie können sich auf den Anwendungsfall und die API konzentrieren, anstatt sich mit der Implementierung zu befassen. Hochspezialisierte Bibliotheksprogramme oder Frameworks werden manchmal als domänenpezifische Sprachen (DSL) bezeichnet, wenn sie gewissermaßen die De-facto-Norm und das gesamte Toolset für eine bestimmte Domäne darstellen. NumPy ist eine DSL für n-dimensionale Arrays in Python. Pandas ist dagegen ein Beispiel einer DSL für Datenrahmen in Python. Um ein:e Expert:in in einer datenintensiven Wissenschaft zu werden, muss man sich in der Regel die wichtigsten Bibliotheken der Domäne aneignen.

Datenbanken

Eine weitere wichtige Anwendungsdomäne, die wir bereits im geschichtlichen Überblick der APIs angesprochen haben, sind Datenbanken. Datenbankverwaltungssysteme sind komplexe Softwarekomponenten, mit denen Benutzer:innen interagieren müssen, und jedes derartige System verfügt über Sprachen bzw. APIs, die genau das tun. Moderne Datenbanken, die in den letzten beiden Jahrzehnten erstellt wurden, weichen von der herkömmlichen Arbeitsweise ab, aber relationale Datenbanken teilen die Eigenschaft, dass sie auf Tabellen mit genau definierten Spaltennamen und -arten basieren. Darüber hinaus haben relationale Datenbanken eine gemeinsame Abfragesprache, nämlich Structured Query Language (SQL). Von SQL gibt es Varianten und Dialekte, aber im Grunde handelt es sich dabei um eine API, die zur Interaktion mit relationalen Datenbanken verwendet wird. Es werden damit auch Tabellen erstellt, geändert und gelöscht, Einträge hinzugefügt oder modifiziert, Daten abgerufen und gefiltert usw. Die interne Struktur von relationalen Datenbanken besticht in ihrer Komplexität, und viele Optimierungen werden durch modernes Datenbankdesign sorgfältig vor dem:der Benutzer:in verborgen. Relationale Datenbanken und ihre Datenstrukturen und Algorithmen zur Datenspeicherung und zum Datenabruf können auf die verschiedensten Arten und Weisen implementiert werden. Für Benutzer:innen bleibt es eine Black Box. Der Datenzugriff spielt sich nur über die domänenpezifische Sprache SQL ab.

Remoteprozeduraaufrufe

Bisher haben wir uns nur mit APIs befasst, die sich auf einen einzelnen Prozess auf einem einzelnen Computer beziehen. In der Regel laufen jedoch mehrere Prozesse auf einem Computer gleichzeitig ab. Dazu müssen diese Prozesse kommunizieren: Bei der gleichzeitigen Ausführung eines Textverarbeitungsprogramms zur Protokollierung einer Besprechung und deren Leitung anhand einer Konferenzsoftware, ist das Betriebssystem intelligent genug, um eine Notiz aus dem Textverarbeitungsprogramm in den Chat der

Videokonferenz zu kopieren. Dies ist eine Form der prozessübergreifenden Kommunikation (IPC). Wenn wir noch einen Schritt weiter gehen, sehen wir, dass ein:e Kolleg:in an einem ganz anderen Computer die Notiz in der eigenen Instanz desselben Videokonferenz-tools sieht, das als Prozess auf seinem:ihrer Computer ausgeführt wird. Wenn Computer so miteinander kommunizieren, ist das kein einfacher Vorgang, der normalerweise auf komplizierten Kommunikationsprotokollen beruht. Ein wichtiger Teil der Kommunikation zwischen mehreren Computern ist das Networking und damit die Art und Weise, wie Computer, z. B. über das Internet, miteinander verbunden sind. Ein Beispiel für IPC zwischen verschiedenen Maschinen aus dem Fachbereich des Distributed Computing ist der Remoteprocedureaufruf (RPC), dem ein Anforderung/Antwort-Protokoll zugrunde liegt. Es gibt zwei Parteien: einen Client und einen Server, die über ein Netz miteinander verbunden sind. Der Client schickt eine Anforderung an den Server zur Ausführung eines Programms mit den festgelegten Parametern.

Der Server führt dieses Programm entsprechend aus und gibt das Ergebnis an den Anrufer, also den Client, zurück. Das schwierige daran ist, dass das Programm selbst über das Netz vom Client zum Server übertragen werden muss und das Ergebnis ebenfalls vom Server zum Client übertragen werden muss. Das heißt, dass alle Objekte im Arbeitsspeicher des Clients, die zum Funktionsaufruf notwendig sind, erst persistiert werden, dann übers Netz geschickt und schließlich vom Server wieder in den Speicher gelesen werden müssen. Viele RPC-Systeme verwenden eine Schnittstellenbeschreibungssprache, um den Vertrag zu beschreiben, der dann zur Generierung von Code sowohl für den Client als auch den Server verwendet werden kann. Ein bekanntes und weit verbreitetes RPC-System ist gRPC von Google, das das Protokollpufferformat von Google als IDC und das Internet-weite Hypertext Transfer-Protokoll (HTTP) zur Netzübertragung verwendet. HTTP ist eines der zentralen Protokolle, auf dem das World Wide Web beruht, und das für die meisten modernen Web-APIs von großer Bedeutung ist.

Web-Schnittstellen

Angesichts der Vielfalt von modernen Webanwendungen und Apps mag es überraschen, dass fast die gesamte Kommunikation über Protokolle wie HTTP läuft. Dieses Protokoll wurde offiziell 1997 eingeführt. HTTP/2 gibt es seit dem Jahr 2015, und der Nachfolger, HTTP/3, wird bereits von einigen Browsern unterstützt (Berners-Lee, 1996). Das moderne Internet gründet auf einer Reihe von Constraints oder Prinzipien, die als Representational State Transfer (**REST**) bezeichnet werden. Anwendungen, die diese Prinzipiensammlung nutzen, werden oft RESTful-Anwendungen oder -Dienste genannt. APIs für RESTful-Anwendungen sind folglich RESTful-APIs. Dabei wird häufig HTTP zur Kommunikation verwendet. Die Entwicklung des HTTP-Protokolls als API ist zwar äußerst interessant, im Rahmen des vorliegenden Studienskripts soll jedoch die Semantik von HTTP und seine praktische Bedeutung im Vordergrund stehen. Im Zentrum von HTTP stehen die Uniform Resource Identifiers (URIs), mit denen sich Ressourcen aus dem Internet eindeutig identifizieren lassen, und Operationen oder Methoden, die HTTP zur genauen Verarbeitung dieser Ressourcen nutzt. Der grundlegende Prozess ist der Aufruf des Clients zur Ausführung einer HTTP-Methode, die der entsprechende Server, der zur URI gehört, ausführt, woraufhin der Server das Ergebnis der Anforderung an den Client zurückgibt. Ein prototypisches Beispiel ist die Eingabe einer Adresse wie google.com in den Browser und die Betätigung der Eingabetaste. In diesem Fall ist der Browser der Client, der eine GET-Anfrage an die

REST

Der Begriff des Representational State Transfer (REST) wurde 2000 im Rahmen einer Dissertation eingeführt und legt eine Reihe von Architektur-Constraints für Webanwendungen, wie die Client-Server-Architektur oder zustandslose Dienste, fest. Anhand dieser Constraints waren Entwickler:innen von Webanwendungen in der Lage, gemeinsame Kommunikationskanäle zu finden, um das gigantische, verteilte Netzwerk, das Internet, aufzubauen.

Server von Google schickt, die mit dem HTML-Code für die Landingpage von Google antworten. Diese wiederum kann vom Browser dargestellt werden. Eine vollständige Liste von HTTP-Methoden sieht wie folgt aus:

- Durch GET wird eine Anfrage nach einer bestimmten Ressource ausgelöst. GET wird nur zum Datenabruf verwendet. Eine GET-Anfrage hat weder auf den Server noch sonstige Auswirkungen.
- HEAD ist identisch zu GET, nur dass kein Nachrichtenrumpf gesendet wird. Wenn GET beispielsweise den HTML-Code für eine Webseite liefert, enthält die entsprechende HEAD-Anforderung nur die Metainformationen der Anforderung (z. B. Informationen zum Server), jedoch nicht den eigentlichen Inhalt. Eine kleine Webanwendung muss mindestens HEAD- und GET-Methoden haben, um als solche zu gelten. Die folgenden Methoden sind optional.
- POST unterscheidet sich dadurch von einer GET-Anfrage, dass es einen Anforderungsrumphat, in dem Nachrichten bereitgestellt und vom Server angenommen und schließlich zur Anforderungsausführung vor Rücksendung zum Client verwendet wird. Ein Beispiel hierfür stellt das Posten eines Beitrags in einem Nachrichtenportal dar. Dabei ist der Text der Nachricht im Anforderungsrumph enthalten.
- PUT hängt in gewisser Weise mit POST zusammen, da es ebenfalls Inhalt im Anforderungsrumph übermittelt. Es besteht jedoch ein semantischer Unterschied, da der Server den Nachrichtenrumpf speichern muss, d. h. der Benutzer:in legt die gesendete Ressource dort ab.
- DELETE löscht eine zuvor gespeicherte Ressource, die durch PUT dort abgelegt wurde.
- PATCH modifiziert eine zuvor gespeicherte Ressource mindestens teilweise.
- OPTIONS gibt die vom Server unterstützte HTTP-Methode an den Client zurück.
- TRACE wird hauptsächlich zum Debuggen verwendet und schickt eingehende Anfragen zurück zum Client, so dass dieser prüfen kann, ob die ursprüngliche Anfrage vom Server modifiziert oder ergänzt wurde.
- CONNECT ist eine Methode, bei der der Client den Server anruft, um mit einem anderen Computer oder über ein anderes Protokoll verbunden zu werden. Dabei fungiert der Server als Proxy. Diese Methode wird oft zur Einrichtung einer sicheren Client-Server-Verbindung über Methoden wie **HTTPS** verwendet, die sicher verschlüsselt sind.

HTTPS

Eine Erweiterung von HTTP, die eine kryptografische, protokollähnliche Transport Layer Security (TLS) bzw. den Vorgänger davon, Secure Sockets Layer (SSL) verwendet, um die Verbindung zwischen den Clients und den Servern zu sichern. Dies soll vor korruptem Verhalten im Internet schützen.

Grafische Benutzeroberflächen

Schließlich können auch grafische Benutzeroberflächen (GUIs) als eine Art API dienen. Die meisten Betriebssysteme verfügen über eine grafische Benutzeroberfläche mit einer grafischen Darstellung der Ordnerstruktur, einem Desktop, einem Mauscursor und anderen Vorzügen. In der Anfangszeit der Informatik gab es lediglich eine Befehlszeilschnittstelle und davor nicht einmal die. Heutzutage sind grafische Benutzeroberflächen für die meisten kommerziellen Anwendungen auf einem Desktop oder Smartphone die Regel. Eine besonders relevante Anwendungsklasse von GUIs im Rahmen der datenintensiven Wissenschaften sind die Integrierten Entwicklungsumgebungen (IDEs), durch die Programmierer:innen ein umfassendes Supportsystem für ihre Arbeit zur Verfügung steht. Tools, wie PyCharm oder Rodeo werden in der Python-Programmierung oder von Datenwissenschaftler:innen, deren Produktivitätsansprüche höher sind, gerne verwendet. Diese IDEs bieten Tools zur Code-Vervollständigung, zur Navigation komplexer Projekte, zur Quellcodesuche, zum Codedebugging oder der Erstellung eines Performanzprofils des Pro-

gramms während der Runtime. In gewisser Hinsicht liefern IDEs APIs für alles, was unmittelbar mit den Programmieraufgaben zusammenhängt. Eine grafische Benutzeroberfläche dieser leistungsstarken Tools ist dafür unerlässlich.

4.2 Designprinzipien für APIs

Wie oben gezeigt wurde, gibt es APIs in verschiedenen Formen und Ausführungen. Nun gilt es herauszufinden, wie eine API erstellt wird. Die wichtigsten Designprinzipien für APIs zu kennen, soll Sie dabei unterstützen, ein Gefühl für gutes und schlechtes Design zu entwickeln. Um es mit einfachen Worten zu sagen: Wenn wir etwas schreiben, vor allem Code, wollen wir verstanden werden. Betrachten wir also zunächst zwei konkrete Beispiele eines Machine-Learning-Softwareworkflows. Das erste Beispiel betrifft Ray, ein vielversprechendes Framework für Distributed Computing für Python, das verschiedene Algorithmen für das so genannte Verstärkungslernen, ein Machine-Learning-Paradigma, unterstützt. Diese Python-Bibliothek wird mit `pip install 'ray[rllib]'` installiert. Um zu Prüfung, ob die Installation funktioniert hat, kann `import ray ray.init()` in eine Python-Sitzung eingegeben werden. Dabei sollten keine Fehler auftreten, d. h., wir sollten ein erstes Beispiel ausführen können, in dem wir folgende Zeile in die Shell eingeben:

```
rllib train --run=PPO --env=CartPole-v0
```

Dabei fällt auf, dass dieses Programm nicht richtig läuft. Es wird durch eine Fehlermeldung in einem Output, das zu lang ist, um es hier aufzuführen, unterbrochen. Nach der mühsamen Suche in ca. 80 Zeilen Code entdecken wir schließlich den Hinweis:

```
ImportError: Could not import tensorflow
```

Der Grund dafür ist, dass Ray auf der von Google herausgegebenen Bibliothek TensorFlow basiert, die zuerst installiert werden muss. Das Problem besteht nicht darin, dass Ray TensorFlow nicht installiert hat, was ohnehin falsch wäre, sondern dass wir unsere Installation verifiziert haben und Ray keine Warnung ausgegeben hat. Unser Setup wurde von Ray problemlos akzeptiert und die Ausführung des Beispiels begann. Zudem ist die Fehlermeldung schwer zu lesen, und besonders für Anfänger:innen ist es schwierig, solch langen Mitteilungen Informationen zu entnehmen. Das Beispiel ist auch daher problematisch, da die Fehlermeldung zu spät im Programmfluss auftrat. Der Fehler tritt auf, wenn Ray das Framework für Distributed Computing erstellt hat, daher erscheint die entsprechende Fehlermeldung zu TensorFlow erst nach etwa 50 Zeilen im Programmoutput. Dies stellt ein Beispiel für schlechtes Design dar. Angenommen, man startet abends ein komplizierteres Machine-Learning-Experiment mit Hunderten von Codezeilen. Am nächsten Morgen sieht man, dass das Programm sofort abgestürzt ist, weil zuvor TensorFlow hätte installiert werden müssen. Dadurch kann es zu enormem Produktivitätsabfall kommen, nur weil bei der Fehlerbehandlung eine fragwürdige Designscheidung getroffen wurde. Betrachten wir zum Vergleich, wie ein anderes Machine-Learning-Framework mit derselben Situation umgeht. Keras ist eine Bibliothek, die häufig in Deep Neural Networks einge-

setzt wird. Wie bei Ray muss man noch nichts über Machine Learning wissen, denn wir betrachten in diesem Abschnitt nur das API-Design. Nach der Installation von Keras durch Eingabe von `pip install keras`, kann diese durch

```
import keras
```

in der interaktiven Python-Sitzung validiert werden. Es erscheint die folgende Fehlermeldung:

```
ImportError: Keras requires TensorFlow 2.2 or higher. Install TensorFlow via `pip install tensorflow`
```

Wie Sie sehen, teilt Keras Ihnen unverzüglich mit, dass etwas fehlgeschlagen ist und schlägt Ihnen darüber hinaus eine konkrete Behebungsaktion vor, ohne dass dafür eine große Menge Textoutput durchsucht werden müsste. Diese Methode zur Fehlerbehandlung ist ein Beispiel für ein ausgezeichnetes API-Design und zeigt, dass gut geschriebene APIs vielen Problemen vorbeugen können. Eine gute API erleichtert nicht nur die Programmierarbeit, sondern fördert auch die Produktivität. Keras wird seit seiner Einführung im Jahr 2015 für sein API-Design gelobt und seit einigen Jahren weniger als Bibliothek, sondern viel mehr als API-Spezifikation verstanden, die Sie implementieren und befolgen können (Sadrach, 2020). Seit dem Release von TensorFlow 2.0, was früher eine eigenständige Bibliothek darstellte, ist Keras nun eine offiziell unterstützte API von TensorFlow (Chollet, 2017). Wir kommen später auf Keras zurück, um noch einige zusätzliche Punkte im Hinblick auf gute Designentscheidungen zu erläutern.

Zen of Python

Einer der Gründe, weshalb die Programmiersprache Python als elegant und geeignet für den Anfang gilt, ist die ihr zugrunde liegende und im sogenannten „Zen of Python“ festgehaltene Designphilosophie. Zen of Python ist eine Auflistung flexibler Prinzipien, die nicht mit einem Regelwerk verwechselt werden sollte. Sie kann in jeder Python-Version durch Eingabe von `import this` in der Python-Sitzung abgerufen werden. Diese Prinzipien werden im Folgenden detailliert behandelt, da sie Aufschluss über gutes API-Design und die dabei auftretenden Schwierigkeiten geben. Am Rande sei Daniel Greenfield bemerkt, der sich über diese Philosophie lustig machte, indem er eine Anti-Zen-Bibliothek namens „That“ ins Leben rief, die zu vermeidende Anti-Pattern aufzeigt. Zusammengefasst beinhaltet Zen of Python die folgenden Aspekte:

- **Schön ist besser als hässlich:** Dieses Prinzip bedeutet, dass schöne Lösungen besser sind als hässliche, was sehr subjektiv sein kann. Der Einsatz von intuitiven, sofort verständlichen Begriffen ist für die Benutzer:innen vorteilhaft.
- **Explizit ist besser als implizit:** Wenn dem:der Benutzer:in Informationen verborgen bleiben, müssen sie möglicherweise mehr über die impliziten Entwicklerregeln wissen, es ist also besser, so explizit wie möglich zu sein.
- **Einfach ist besser als komplex, komplex ist besser als kompliziert:** Das Sparsamkeitsprinzip von Python, vergleichbar mit Ockhams Rasiermesser, besagt, dass Komplexität, wenn möglich, vermieden werden sollte. Wenn Komplexität notwendig ist, sollte es wenigstens nicht kompliziert werden.

- **Flach ist besser als verschachtelt, sparsam ist besser als dicht:** Datenstrukturen und -entwürfe sollten nicht zu sehr miteinander verwickelt oder verschachtelt sein. Ein flaches Python-Dictionary mit Eigenschaften wird einer verschachtelten Lösung vorgezogen. Eine Codezeile sollte nicht zu voll gepackt werden, d. h., Informationsdichte gilt es zu vermeiden.
- **Lesbarkeit zählt:** Code, den man leichter nachvollziehen kann, liest sich natürlich besser und kann leichter langfristig gepflegt werden.
- **Sonderfälle sind nicht besonders genug, um gegen die Regeln zu verstößen:** Dennoch steht Durchführbarkeit über Regeltreue. Auf Metaebene zeigt uns dieses äußerst interessante Zen-Prinzip, dass die Prinzipien nicht ohne Vorbehalte verwendet werden können. In der Regel sollten Sonderfälle das Gesamtdesign nicht ruinieren. Manchmal sind es aber genau diese Ausnahmen, die unbedingt notwendig sind.
- **Fehler sollten nie unbemerkt passieren, außer sie werden ausdrücklich ausgeblendet:** Wenn Fehler schnell auftreten, müssen Benutzer:innen nie raten, warum das Programm nicht funktioniert. Dadurch wird der Programmfluss womöglich komplizierter, es ist dennoch eine gute Designentscheidung. In seltenen Fällen und wenn dadurch keine anderen Konsequenzen entstehen, kann eine Fehlermeldung durchaus übergangen werden.
- **Bei Mehrdeutigkeit sollte der Versuchung, zu raten, widerstanden werden:** So sollte man Mehrdeutigkeit vermeiden und dafür so explizit und klar wie möglich sein.
- **Es sollte eine — und vorzugsweise nur eine — offensichtliche Aktionsmöglichkeit geben, obwohl diese auf den ersten Blick womöglich nicht offensichtlich ist, sofern man kein:e Niederländer:in ist:** Dieser Verweis auf den niederländischen Python-Entwickler, Guido Van Rossum, stellt wahrscheinlich das strittigste Prinzip der gesamten Liste dar. Python-Programmierer:innen nennen das „pythonisch.“ Oft gibt es keinen Zweifel an einer pythonischen Lösung und es herrscht Einigkeit in Bezug auf die meisten Muster. In manchen Fällen ist es jedoch nicht klar, ob ein Ansatz tatsächlich besser ist als ein anderer. Dennoch sollte es das Ziel sein, den Code so klar zu schreiben, dass damit keine Fehler passieren können.
- **Jetzt ist besser als nie:** Obwohl nie bisweilen besser ist als gerade jetzt. Beim Design sollte man den Prozess nicht hinauszögern und gleich eine Lösung finden, selbst wenn diese nicht perfekt ist. Sie kann später durch Iteration verbessert werden. Im Notfall sollte dennoch kein Weg eingeschlagen werden, der später zu Komplikationen führen könnte.
- **Wenn die Implementierung schwer zu erklären ist, ist sie schlecht:** Wenn die Implementierung leicht zu erklären ist, könnte sie gut sein. Diese etwas pessimistische Ansicht des Programmierens weist noch einmal auf Schönheit und Verständlichkeit hin. Nicht alles, was sich leicht erklären lässt, ist gut, aber wenn Sie es nicht erklären können, ist es definitiv schlecht.
- **Namespaces sind eine hervorragende Idee — davon brauchen wir viele!** Python setzt Namespaces häufig zur Modularisierung und Struktur des Codes ein. Möchten Sie zum Beispiel einen Zoo mit vielen Tieren in Python implementieren, so könnte die Klasse Elephant mit dem Befehl `from zoo.animals.mammals import Elephant` statt durch `from zoo import Elephant` importiert werden, da durch Letzteres möglicherweise zu viele Begriffe im Namespace eingeführt würden.

Trennung von Schnittstelle und Implementierungen

Ein allgemein gültiges Prinzip beim Schnittstellendesign ist die Trennung von Schnittstellen und ihren Implementierungen. Streng genommen handelt es sich bei einer API, den besprochenen Definitionen und Eigenschaften zufolge, schlicht um eine Schnittstelle, deren Zweck es ist, dass die Benutzer:innen den Implementierungsdetails keine Beachtung schenken müssen. In der Praxis ist es jedoch sinnvoll, sich dieses Prinzip hin und wieder zu vergegenwärtigen. Die Trennung von Schnittstellen und Implementierungen lässt sich im Rahmen der „**Trennung von Zuständigkeiten**“ verstehen. Stark tipisierte Sprachen wie Java oder Scala haben dedizierte Schnittstellenklassen, durch die diese Praktik unterstützt wird. In Java kann eine Schnittstelle keine konkrete Implementierung haben, da es das Design nicht zulässt. Wenn man mit der Funktionalität, wie auf der Schnittstelle arrangiert, arbeiten möchte, muss zuerst eine Schnittstellenimplementierung geschaffen werden, in dem eine Unterklasse geschrieben wird. Grundsätzlich können die beiden Programmiersprachen Java und Scala, die syntaktisch sehr unterschiedlich sind, in kompatiblem Bytecode auf der Java Virtual Machine (JVM) kompiliert werden. So können Java-Klassen in den Scala-Code importiert und die Software darauf aufgebaut werden. In Python gibt es die Idee einer Schnittstelle in der Niedrigsprache nicht. Eine strikte Trennung von Schnittstellen und konkreten Implementierungen werden als Anti-Pattern (oder unpythonisch) angesehen. Dadurch wird das vorgestellte Prinzip jedoch nicht vollkommen außer Kraft gesetzt. So ist es dennoch notwendig, in Python die Objekte, mit denen Benutzer:innen interagieren, die öffentliche API des Codes, den restlichen Code und die Implementierung gut zu durchdenken. Wenn Sie die öffentlich zugängliche API als Oberfläche betrachten, auf der die meisten Benutzer:innen arbeiten, so stellt sie für größere Projekte lediglich die Spitze des Eisbergs dar, weshalb gutes Schnittstellendesign umso bedeutsamer wird.

Abstraktionen

Bei der Erstellung von APIs ist es wichtig, den richtigen Abstraktionsgrad zu finden, d. h. die richtigen Begriffe, die gut zum Endprodukt passen. Abstraktionen sollten natürlich sein und dem:der Benutzer:in die Interaktion erleichtern. Um es mit den Worten des berühmten Informatikers Edsger Dijkstra zu sagen „abstrakt ist etwas ganz anderes als vage ... Der Zweck der Abstraktion besteht nicht darin, vage zu sein, sondern eine neue semantische Ebene zu schaffen, auf der man absolut präzise sein kann“ (Misa & Phillip, 2010, S.1, vom Autor übersetzt). Die Keras-Bibliothek kommt beispielsweise mit sehr wenigen höheren Begriffen aus. Um die Eleganz des Keras-Ansatzes zu verstehen, muss man wissen, dass Deep Neural Networks, für die Keras entwickelt wurde, Machine-Learning-Modelle aus Ebenen von relativ einfachen Compute-Anweisungen sind. Die einfachsten Modelle sind sequenziell aufgebaut, das heißt, dass eine Ebene der nächsten folgt. Die einfachste Ebene eines neuronalen Netzes ist wahrscheinlich eine Ebene mit dichten Verbindungen; alle Neuronen (ein Ausdruck aus der Neurobiologie für Nervenzellen) in einer Ebene sind mit Neuronen der nächsten verbunden. Mithilfe der obigen Informationen können Sie folgende Keras-Modellspezifikation verstehen, selbst wenn Sie noch nie etwas von Deep Neural Networks gehört haben:

Trennung von Zuständigkeiten

Ein gebräuchliches Prinzip in der Informatik, bei dem die Zuständigkeiten getrennt werden, wobei die Software so konzipiert ist, dass jeder Teil oder jedes Modul für einen bestimmten Teilaспект zuständig ist. Zuständigkeiten sollten nie auf zwei oder mehr Module verteilt sein.

Code

```
import keras
from keras import layers
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

Um also ein sequenzielles Netz mit Keras zu bauen, fügen Sie sequenziell so viele Ebenen hinzu, wie Sie möchten. Die Einzelheiten dieses Modells sind hier nicht von Belang, es sei jedoch darauf hingewiesen, dass sich der Code beinah wie einfaches Englisch liest und mit genau der gleichen, hochgradigen Abstraktion arbeitet, die wir gerade erläutert haben: durch Modelle und Ebenen. Das mag offensichtlich erscheinen, das ist es aber keineswegs. Vor Keras verwendete keines der Deep-Learning-Frameworks einen ähnlich simplen und intuitiven Programmierstil, wodurch der Bereich des Deep Learning für Fachkräfte unzugänglicher war. Im Jahr 2020 hat sich die Situation zum Besseren gewandelt, jedoch sollte der Einfluss von gut geschriebenen APIs wie Keras auf den Trend des Deep Learnings und des Machine Learnings nicht unterschätzt werden. Bekanntmaßen sind alle Abstraktionen undicht (Spolsky, 2002). Das bedeutet, dass es, ganz gleich wie gut eine Abstraktion in der Informatik durchdacht ist, immer spezielle Fälle geben wird, in denen man genauer nachsehen oder bestehende Regeln brechen muss. Dies spiegelt die Unordnung in der Welt wider und verweist auf die Tatsache, dass Abstraktionen nur ein Versuch der Einteilung von Dingen nach Kategorien sind, ein Prozess, der aus verschiedenen Gründen scheitern kann. Diese Aussage ist zwar wahr, sie spielt jedoch in der Praxis kaum eine Rolle: Mit guten Abstraktionen kann man viel erreichen, während schlechte Entscheidungen oft zu miserablen Ergebnissen führen. Eine weitere Sichtweise über Abstraktionen, insbesondere in Bezug auf undichte Abstraktionen, besteht darin, dass Sie verschiedene Abstraktionsebenen benötigen könnten. Mitunter reicht die höchste Abstraktionsebene aus, aber manchmal muss man einige Ebenen tiefer gehen, um den Code dem jeweiligen Anwendungsfall anzupassen.

Fehlerbehandlung

Am Beispiel der Ray- und Keras-Python-Bibliotheken haben wir bereits gute und schlechte Fehlerbehandlung gesehen, doch lohnt sich die Generalisierung und Zusammenfassung einiger wichtiger Punkte, die es zu beachten gilt:

- Wenn Fehler auftauchen, ist es am besten, wenn das so früh und schnell passiert, wie möglich. Dadurch wird maximale Transparenz erreicht und schwerwiegende Fehler später im Programm, die oft schwieriger zu analysieren und beheben sind, vermieden.
- Fehlermeldungen sollten für Menschen lesbar und klar geschrieben sein. Sie sollten dem:der Benutzer:in keine lokale, schwer verständliche Meldung zum Fehler vorlegen, sondern den Kontext des Fehlers und Abhilfe beschreiben.
- Auch sollten Fehlermeldungen verschiedener Art vorbereitet werden, die beispielsweise nützliche Warnungen zur Einstellung veralteter Funktionen, die im nächsten Release womöglich nicht mehr vorhanden sind, geben.

Konvention und Konfiguration

Laut dem Informatiker Phil Karlton „gibt es in der Informatik nur zwei Dinge, die schwierig sind: die Cacheinvalidierung und die Benennung von Dingen“ (Fowler, 2009, S. 1, vom Autor übersetzt). Trotz eines Augenzwinkerns ist diese Aussage nicht gänzlich falsch. Auf das API-Design trifft hauptsächlich ein Teil des Zitats zu, nämlich, dass Dinge schwer zu benennen sind. Denn es erweist sich überraschenderweise als schwierig, gute Namenskonventionen zu finden und einige Sprachen, wie Java, tendieren zu sehr langen Klassennamen. Beispielhaft soll hier `AbstractTransactionalDataSource-SpringContextTests` aus dem Spring-Framework angeführt werden, um zu verdeutlichen, was es zu vermeiden gilt: Das Beispiel deutet entweder auf einen Mangel an begrifflicher Klarheit in der Abstraktion hin oder darauf, dass die Klasse mit zu viel belegt ist (womit auch das Prinzip der Trennung von Zuständigkeiten verletzt wäre). Klare und präzise Namensgebung von Klassen und Funktionen sollte beim Softwaredesign stets berücksichtigt werden. Neben den Namenskonventionen spielen auch andere Konventionen eine Rolle. In Python können beispielsweise auch Standardargumente in die Funktionssignatur geschrieben werden. Davon sollte man immer dann Gebrauch machen, wenn sinnvolle Standards vorhanden sind. Die meisten Benutzer:innen sind keine Power-User und interessieren sich nicht für jeden einzelnen Parameter, der angepasst werden kann. Es ist jedoch besser, in den Funktionsparametern viele konfigurierbare Parameter zu haben, als sie von Benutzer:innen durch Code modifizieren zu lassen. Im folgenden Python-Beispiel gibt es eine einfache Klasse mit einem Bucket, der eine bestimmte Größe hat und mit etwas gefüllt werden kann. So kann eine Python-Klasse dafür dargestellt werden:

Code

```
class Bucket:  
    def __init__(self, size=10):  
        self.size = size  
  
    def update_size(self, size):  
        self.size = size
```

Die Standardgröße für den Bucket ist 10 (die Bedeutung oder die Einheit spielt hier keine Rolle), also können Benutzer:innen Buckets mit folgender Konvention erstellen.

```
bucket = Bucket()
```

Sie können auch die Größe des Buckets konfigurieren:

```
bucket = Bucket(size=20)
```

Außerdem könnten sie zuerst einen Bucket erstellen, dessen Größe sie anschließend durch folgenden Code anpassen können (allerdings ist dies ein fragwürdiger Vorgang)

Code

```
bucket = Bucket()  
bucket.set_size(20)
```

Beim Design von APIs gilt die Faustregel „Konvention geht über Konfiguration, Konfiguration geht über Code.“ Es empfiehlt sich, so viele gute Standardwerte wie möglich zu liefern und deren Konfigurierung so einfach wie möglich zu machen.

Benutzererlebnis

Viele der in diesem Abschnitt besprochenen Prinzipien beruhen auf dem Einsatz von gesundem Menschenverstand auf das Fachgebiet der Softwareentwicklung. Denn bei guten APIs oder bei gutem Design im Allgemeinen geht es stets darum, sich in Benutzer:innen hineinzuversetzen und die kognitiven Faktoren zu berücksichtigen, die bei der Erstellung von Produkten für Menschen eine Rolle spielen. Mit anderen Worten, auf das Benutzer:innenerlebnis kommt es an – auch User Experience oder, wenn es um die Beziehung zwischen Ingenieur:in und Code geht, auch speziell Entwickler:innenerlebnis. Das Cognitive Dimensions Framework (Clarke, 2004) führt insgesamt zwölf Punkte auf, die beschreiben, wie Informatiker:innen mit APIs arbeiten und was sie von APIs erwarten:

- **Abstraktionsebene:** Was ist die maximale Anzahl und die Mindestanzahl der Abstraktionen in der API und was liegt dazwischen? In welcher Ebene sollten Benutzer:innen hauptsächlich arbeiten?
- **Lernstil:** Welche Anforderungen bestehen, wenn man die API erlernen möchte? Ist die Lernkurve steil? Was ist die Entwickler:innenzielgruppe und über welches Vorwissen verfügen sie?
- **Arbeitsrahmen:** Wie sieht die kognitive Beanspruchung aus bzw. welche Begriffe muss man verstehen, um die API verstehen und mit ihr effektiv arbeiten zu können?
- **Arbeitsschritte in Einheiten:** Wie viel wird von Benutzer:innenn in den einzelnen Schritten verlangt? Ist die API in dieser Hinsicht besonders anspruchsvoll?
- **Progressive Bewertung:** Können Codeteile progressiv erstellt werden und wird dabei Feedback angeboten? Oder müssen im Extremfall zum Konzeptnachweis alle Teile zusammengeführt werden?
- **Frühzeitige Festlegung:** Wie viele komplexe nachfolgende Entscheidungen muss man bei einer bestimmten Entscheidung bereits mitberücksichtigen?
- **Verständlichkeit:** Wie gut kann man die API allein erlernen? Ist sie gut dokumentiert und kann man die Komponenten intuitiv nachvollziehen und zusammenbauen?
- **API-Ausführung:** Wie gut entspricht die API den jeweiligen Bedürfnissen? Kann sie flexibel konfiguriert werden oder muss sie stark an die Bedürfnisse der Benutzer:innen angepasst werden?
- **Schwerfälligkeit der API:** Ist es leicht, die API an den Anwendungsfall anzupassen und zu erweitern? Wird die Sprache der angestrebten Entwickler:innencommunity verwendet?
- **Konsequenter Aufbau:** Können aus einem Teil der API Rückschlüsse auf die Funktionalität eines verwandten API-Teils gezogen werden? Wurden die Bausteine gut und konsequent ausgewählt?
- **Ausdruckskraft der Rollen:** Entsprechen die für die API gewählten Abstraktionen dem kognitiven Modell der Komponenten im gesamten Programmfluss?
- **Anpassung an das Fachgebiet:** Alle APIs werden in einem Kontext, für ein bestimmtes Fachgebiet geschrieben. Wie gut drückt die API die im Fachgebiet vorhandenen Modelle aus?

Dokumentation

Abschließend sei noch bemerkt, dass gute Dokumentation für erfolgreiche APIs mit vielen Benutzer:innen dringend erforderlich ist. Folgende Aspekte sollten stets abgedeckt werden:

- Dokumentation des Codes,
- Dokumentation der Schnittstelle, Eingabeparameter und Rückgabewerte sowie
- Dokumentation der gedachten Verwendung der API.

4.3 Erstellen einer Python-Bibliothek

Nachdem wir einige API-Musterbeispiele und Designprinzipien betrachtet haben, können wir das Gelernte bei der Erstellung einer Python-Bibliothek anwenden. Dabei sollen nicht unbedingt die Implementierungsdetails im Zentrum stehen, sondern viel mehr die API. Der Code, den wir für diese API schreiben, wird relativ kompliziert ausfallen, so dass wir zeigen können, wie gutes Design einer API diese Komplexität verbergen kann. Hier ist unser konkretes Beispiel: Stellen Sie sich vor, dass Sie eine Firma führen, die auf die Analyse großer Korpora aus Textdokumenten spezialisiert ist. Zudem haben Sie einen proprietären Algorithmus, der umfangreiche Analysen schneller als jeder andere ausführen kann. Sie möchten den Benutzer:innen diesen Algorithmus anbieten, um dem Bedarf gerecht zu werden und damit Geld zu verdienen. Der erste Schritt ist die Erstellung einer Python-Bibliothek, in der der Algorithmus sauber gespeichert werden kann und anhand dieser er für Benutzer:innen zugänglich wird, ohne dass der Algorithmus selbst, der vertrauliches Wissen enthält, geteilt wird. Zur weiteren Einschränkung arbeiten wir nur eine Funktion aus, nämlich die Zählung aller Vorkommnisse der Wörter in den verschiedenen Dokumenten.

Ein Algorithmus für die Zusammenfassung von Dokumenten

Wir beginnen mit der Implementierung des Kernalgorithmus. Dies ist eine Miniversion des Programmierungsmusters, MapReduce, dessen Einführung ein Meilenstein im Distributed Computing war und einer von vielen Beiträgen Googles zur Hochleistungsinformatik (Dean & Ghemawat, 2004). Viele kommerziell erfolgreichen Technologien im Big Data-Bereich, wie Hadoop, gründen auf diesem Modell. Es beruht vereinfacht gesagt auf drei Schritten:

1. Einen Stapel Dokumente nehmen und die wichtigen Elemente (z. B. die Wörter im Dokument) transformieren oder zuordnen (von Englisch „map“) nach einer von Benutzer:innen eingegebenen Funktion. Bei dieser Stufe werden Schlüssel-Wert-Paare zurückgegeben, bei denen die Schlüssel die wichtigen Dokumentelemente sind

und der Wert die Einheit, die berechnet werden soll. Wir wollen Wörter zählen, daher sollte jedes Wort x in einem Dokument zu dem Schlüssel-Wert-Paar $(x, 1)$, führen, bei dem „1“ aussagt, dass das Wort einmal gezählt wird.

2. Alle Ausgabepaare aus dem letzten Schritt sammeln und gruppieren. Angenommen das Wort x ist vier Mal im Dokument enthalten. Der Gruppierungsschritt würde dann zum Wert x führen: $[1, 1, 1, 1]$.
3. Der letzte Schritt ist die Aggregation bzw. Reduzierung der Elemente aus dem letzten Schritt. In unserem Fall bedeutet das, dass die Werte addiert werden sollten, um eine Gesamtsumme zu erhalten. Für das Beispiel des Wortes x erhalten wir daher das Ergebnis $x : 4$.

Dem englischen Namen des Musters MapReduce kann man entnehmen, dass es aus den Schritten 1 und 3 entstanden ist, da zunächst zugeordnet („map“) und anschließend reduziert („reduce“) wird. Aber der zweite Schritt ist ebenfalls wichtig. Die drei Schritte mögen einfach aussehen, doch verfügen sie über das Potenzial, dass sie über Hunderte von Maschinen massiv parallel ausgeführt werden können. Als nächstes erstellen wir eine simple Implementierung dieses Algorithmus für unseren Anwendungsfall, also für die Wortzählung. Wir beginnen mit der Implementierung der drei Phasen. Als erstes definieren wir die Zuordnungsfunktion, die $(\text{word}, 1)$ als Schlüssel-Wert-Paar für jedes Wort im Text zurückgibt:

Code

```
def map_function(text):
    for word in text.lower().split():
        yield word, 1
```

Als zweites wenden wir die Zuordnungsfunktion auf alle Elemente in unserem Datenset data (eine Liste an Textdaten) an:

Code

```
def apply_map(data):
    map_results = []
    for element in data:
        for map_result in map_function(element):
            map_results.append(map_result)
    return map_results
```

Zur Gruppierung der Schlüssel-Wert-Paare, die bei diesem Zuordnungsschritt durch den Schlüssel entstanden, machen wir Folgendes:

Code

```
def group_function(map_results):
    group_results = dict()
    for key, value in map_results:
        if key not in group_results:
```

```

        group_results[key] = []
        group_results[key].append(value)
    return group_results

```

Zum Schluss definieren wir eine Reduzierungsfunktion, die die gezählten Wörter zusammenzählt:

Code

```

def reduce_function(key, values):
    total = 0
    for count in values:
        total += count
    return key, total

```

Wie bei der Zuordnungsfunktion müssen wir auch die Reduzierungsfunktion auf alle Schlüssel-Wert-Paare aus der Gruppierungsphase anwenden:

Code

```

def apply_reduce(group_results):
    reduce_results = dict()
    for key, values in group_results.items():
        _, count = reduce_function(key, values)
        reduce_results[key] = count
    return reduce_results

```

Zur Probe erstellen wir nun einen Testdatensatz, um dieses Programm auszuführen:

Code

```

text_1 = "Lorem ipsum dolor sit amet, consetetur et sadipscing elitr."
text_1b = "Lorem ipsum dolor sit amet, consetetur et sadipscing elitr."
text_2 = "At vero lorem et accusam et justo duo ipsum et ea rebum."
text_2b = "At vero lorem et accusam et justo duo ipsum et ea rebum.
At vero lorem et accusam et justo duo ipsum et ea rebum."
data_set = [text_1, text_1b, text_2, text_2b]

```

Wenn wir alle MapReduce-Schritte einen nach dem anderen ausführen, erhalten wir das folgende Output:

Code

```

map_results = apply_map(data_set) # 1. stage
print(map_results) # Output: [('lorem', 1), ('ipsum', 1), ('dolor', 1), ...]
group_results = group_function(map_results) # 2. stage
print(group_results) # Output: {'lorem': [1, 1, 1, 1, 1], 'ipsum':
[1, 1, 1, 1, 1], 'dolor': [1, 1] ...}
reduce_results = apply_reduce(group_results) # 3. stage
print(reduce_results) # Output: {'lorem': 5, 'ipsum': 5, 'dolor': 2 ...}

```

Nun haben wir gesehen, dass unser Code wie angenommen funktioniert, aber es fehlen natürlich noch wichtige Aspekte. Wir haben eine Miniversion eines mächtigen, kommerziell erfolgreichen Algorithmus implementiert und ihn auf einen konkreten Anwendungsfall angewendet. Wir haben jedoch noch keine API für den Algorithmus. Das geistige Eigentum darf auf keinen Fall an die Öffentlichkeit gelangen, daher müssen wir die Implementierungsdetails sorgfältig verbergen und Benutzer:innen eine einfache Schnittstelle zur Analyse ihrer Dokumente zur Verfügung stellen. Aber wie? Als erstes müssen wir die Spezifikationen der API genau untersuchen, um zu verstehen, was genau die Benutzer:innen brauchen. Wenn wir ein Minimum Viable Product (MVP), d. h. ein Produkt, das mit der geringstmöglichen Funktionalität ausgestattet ist, erstellen möchten, können wir mit folgender Funktionalität in Python beginnen, die wir den Benutzer:innen liefern können:

Code

```
def count_words_naive(corpus):
    map_results = apply_map(data_set)
    group_results = group_function(map_results)
    reduce_results = apply_reduce(group_results)
    return reduce_results
```

Zum Schutz des geistigen Eigentums bekommen die Benutzer:innen nur Zugriff auf `count_words_naive`, nicht jedoch auf den implementierten Code. Aus der ersten Interviewrunde mit potenziellen Kund:innen entnehmen wir folgendes Feedback, das zur Ansatzverfeinerung verwendet werden kann:

- Viele Benutzer:innen haben nicht verstanden, wie diese Funktion verwendet werden soll. Das Eingabeargument `corpus` ist nicht dokumentiert und es ist daher nicht klar, wie es verwendet werden soll.
- Einige Benutzer:innen konnten sich erschließen, dass die API eine Stringliste in Python verwendet, die die Dokumente, die analysiert werden sollen, darstellt. Sie hatten jedoch das Problem, dass die Funktion manchmal zu Fehlern führte, und sie konnten nicht nachvollziehen, warum.
- Benutzer:innen, die das Ergebnis manchmal erzielen konnten, wollten eine etwas andere Lösung: Sie waren an Wörtern mit einer bestimmten Mindestzahl an Vorkommnissen im Korpus interessiert (sonst ist die Ausgabe zu groß).
- Einige Benutzer:innen kannten Python nicht gut genug und wollten lediglich eine Dateiliste statt der einzelnen Dokumente in Python laden.

Um dieses Feedback umzusetzen, entwerfen wir zunächst eine neue Backendfunktion, mit der die Wörter, deren Vorkommensanzahl zu niedrig ist, ausgefiltert werden:

Code

```
def filter_function(results, min_occurrences=4):
    return {k: v for k, v in results.items() \
            if v >= min_occurrences}
```

Mit dieser Filterfunktion können wir nun eine bessere Schnittstelle erstellen, indem wir einige der besprochenen Designprinzipien umsetzen, nämlich saubere und intuitive Fehlerbehandlung, gute Dokumentation, gute Konventionen und Konfigurationen. Dabei bauen wir sogar das Typüberprüfungsmodul typing von Python für Eingabeparameter und Rückgabearten ein:

Code

```
from typing import List, Optional, Dict
import warnings

def count_words(corpus: List[str], filter: Optional[int] = None)
-> Dict[str, int]:
    """Count all words in a corpus of documents.
    :param corpus: A list of strings, where each string
                   contains the text of a document.
    :param filter: int or None. If None, don't filter
                   the result. Else return words
                   with at least 'filter' occurrences
                   in the corpus.
    :return: A dictionary of words and their respective
            counts, filtered by count as specified.
    """
    assert type(corpus) is list, "The corpus to analyse needs to be
list of strings" map_results = apply_map(corpus)
group_results = group_function(map_results) reduce_results =
apply_reduce(group_results) if filter is None:
    return reduce_results
else:
    assert type(filter) is int, f"The 'filter' argument
needs to be a Python int,
you specified: '{filter}' which is of type {type(filter)}"
    return filter_function(reduce_results, min_occurrences=filter)
```

Obwohl dies nun nach mehr Code aussieht, als wir in unserer einfachen Definition festgehalten haben, brauchen wir die meisten Zeilen für die Typüberprüfung, die Dokumentation und zur Fehlererkennung. Das ist ein gutes Zeichen und ausgereifte Schnittstellen sehen in etwa so aus. Um die Anforderung eines API-Aufrufs zu erfüllen, der mit Dateien statt mit Strings aus dem Speicher funktioniert, können wir eine neue Schnittstelle definieren, die die oben erstellte Funktion `count_words` intern aufruft. Die Wiederverwendung von Funktionsaufrufen, anstatt eine separate Codebasis zu erstellen, ist eine gute Designpraktik.

Code

```
def count_words_from_files(files, skip_corrupted_files=False, filter=None):
    """Count all words in a corpus of documents provided as files
    :param files: A list of file names (str) on your local file system.
    The files contain the text you want to count words for.
```

```

:param skip_corrupted_files: boolean. If True, ignore all files that
    can't be read, otherwise abort.
...
"""
corpus = []
for file_name in files:
    with open(file_name, 'r') as f:
        try:
            text = f.read()
            corpus.append(text)
        except:
            msg = f"The file {file_name}"
            cannot be read. Remove it
            from the corpus.
            if skip_corrupted_files:
                warnings.warn(msg)
            else:
                raise ValueError(msg)
return count_words(corpus, filter) # reuse function

```

In diesem Code steckt eine ganze Menge. Erstens unterscheiden sich die Argumente `count_words_from_files` und `count_words` in vielerlei Hinsicht. Der Parameter „`files`“ erfordert zwar immer noch eine Liste an Strings in Python, aber seine Bedeutung hat sich geändert. Wie in der Dokumentation erklärt, ist „`files`“ nun eine Liste von Pfaden auf dem lokalen Rechner, die analysiert werden sollen. Der Parameter „`filter`“ ist unverändert, aber es gibt auch einen neuen: `skip_corrupted_files`. Mit diesem Parameter wird der Fall abgedeckt, dass nicht alle Dateien auf dem System geöffnet werden können oder Textdaten enthalten.

`skip_corrupted_files` ist standardmäßig falsch, d. h. das Programm unterbricht sofort, wenn eine Datei nicht lesbar ist. Dieser Standard ist gut, denn es ist sicherer, als das Ereignis zu überspringen. Benutzer:innen möchten diese Funktion auf eigenes Risiko vielleicht abstellen. Daher machen wir diesen Parameter verfügbar, so dass sie ihn wie gewünscht konfigurieren können. Der Code an sich hat einen einfachen Fluss. Es findet eine Iteration über die Dateien, die gelesen werden, statt. Bei Erfolg wird das Ergebnis an die zuvor definierte Funktion `count_words` ausgegeben.



ZUSAMMENFASSUNG

In dieser Lektion wurden die Grundlagen, das Design und die Implementierung von Anwendungsprogrammierschnittstellen (APIs) unter Berücksichtigung datenintensiver Projekte behandelt. Alle Programmierbeispiele in dieser Lektion wurden in Python durchgeführt, da es leicht ist, in Python Prototypen zu erstellen und da es derzeit die wichtigste Sprache für Projekte in den Datenwissenschaften und verwandten Gebieten darstellt. Im ersten Lernzyklus wurde auf die grundlegende Terminologie

für APIs eingegangen und darauf, welche Rolle verschiedene Abstraktionsebenen bei der Arbeit an Computern spielen, vom Betriebssystem und Hardwarezugriff über moderne Web-Schnittstellen. Diese API-Ebenen decken eine breite Klasse an wichtigen Einstiegspunkten für die Anwendungsentwicklung ab. Dazu gehören der Datenbankzugriff, die Programmiersprachen und die Verwendung der bevorzugten Toolkette in datenintensiven Fachgebieten. Von nun an betrachten wir APIs als Spezifikationen zwischen API-Designer:in und Benutzer:in, als einen Vertrag, an den alle Parteien gebunden sind. Darüber hinaus konzentrierte sich dieser Abschnitt auf die Verwendung von Bibliotheken und Frameworks und die wichtigsten Bausteine von RESTful Webanwendungen. Anschließend haben wir uns angesehen, was zur API-Entwicklung gehört, indem wir die wichtigsten Designprinzipien sowie gute und schlechte Praktiken beleuchtet haben. Weitere Elemente, wie die Trennung der Zuständigkeiten, die gute Fehlerbehandlung, ausreichende Dokumentation, gute Abstraktionen mit wenig Durchlässigkeit sowie Konventionen und Konfigurationen, die die Navigation der APIs in der Praxis vereinfachen, wurden behandelt. Im darauffolgenden Abschnitt befassten wir uns mit einem Anwendungsfall: einer Bibliothek in Python, die Benutzern zur Verfügung gestellt werden soll. Ein komplexer, bereits bestehender Algorithmus, mit dem Wörter in Dokumenten gezählt werden können, wurde in eine neue, für Benutzer:innen sehr kleine API eingebaut. Es wurde weiterhin gezeigt, wie für Benutzer:innen uninteressante Funktionen effektiv versteckt, Code klar dokumentiert, Typüberprüfung von Eingabe- und Ausgabeargumenten verwendet und Fehlerhandhabung zur Aufdeckung von ungewolltem Verhalten eingesetzt werden. Sie sollten nun in der Lage sein, die Stärken und Schwächen von APIs zu beurteilen, sie effektiv zu erstellen und eigene Bibliotheken und REST-Schnittstellen in Python einzurichten.

LEKTION 5

VOM MODELL ZUR PRODUKTION

LERNZIELE

Nach der Bearbeitung dieser Lektion werden Sie in der Lage sein, ...

- die Prozessdetails zur Entwicklung eines Machine-Learning-Modells zu beschreiben und wie es sich vom Standardprozess für Softwareentwicklung unterscheidet.
- den Lebenszyklus eines Modells in der Produktion; Aktualisierung, Monitoring, Integritätsprüfungen, Leistung und Skalierung zu verstehen.
- die Einzelheiten der Modellbereitstellung zu nennen und wie man es verfügbar hält.
- den Zweck von MLOps und DataOps zu verstehen.
- zu erläutern, inwiefern End-to-End-Cloud Services umfassende Lösungen für alle oben genannten Themen anbieten, insbesondere der Amazon SageMaker.

5. VOM MODELL ZUR PRODUKTION

Einführung

Eine Machine-Learning-Lösung bietet nichtdeterministische Vorhersagen auf der Basis von Inputdaten an. Dazu wird ein iterativer und explorativer Prozess angewandt. In dieser Lektion wird zunächst beleuchtet, inwiefern Organisationen Machine Learning nutzen können. Anschließend werden verfügbare Daten gesammelt, analysiert und es wird eine Hypothese über die Art der Daten, das Problem und die Lösung aufgestellt, welche daraufhin sowohl visuell als auch programmatisch anhand von Machine-Learning-Modellen getestet werden soll. Sobald ein Lösungsvorschlag ermittelt ist, wird dieser in die bestehende Infrastruktur integriert bzw. wird eine Software entwickelt, um die Lösung einzusetzen. Im Anschluss daran sollen Ähnlichkeiten und Unterschiede zwischen dem Entwicklungsprozess für Machine-Learning-Lösungen und herkömmliche Softwarelösungen untersucht werden. Eine Machine-Learning-Lösung in Betrieb zu nehmen, birgt Herausforderungen aufgrund der nichtdeterministischen Arbeitsweise des Systems, der jeweiligen Besonderheiten von Teams, die solche Lösungen entwickeln, der inhärenten Abhängigkeit von Trainingsdaten und aufgrund der Daten, auf die die Lösung angewandt wird. Die besonderen Herausforderungen von Versionsverwaltung, Testen und Monitoring, die durch die speziellen Eigenschaften von Machine-Learning-Lösungen entstehen, werden besprochen. In einem späteren Abschnitt werden Tools behandelt, mit denen einige dieser Schwierigkeiten in der Produktionsumgebung angegangen werden können. Dabei wird das MLOps-Modell eingeführt. Außerdem werden der Einsatz und die Erweiterung von Modellen der kontinuierlichen Integration (CI) und Lieferung (CL) in Machine-Learning-Lösungen sowie das kontinuierliche Training (CT) thematisiert. Schließlich werden MLFlow, Kubeflow und Michelangelo näher erläutert. In einem letzten Schritt wird untersucht, welche Lösungen bestehende Cloudanbieter für die speziellen Herausforderungen von ML anbieten, wobei der Amazon SageMaker genauer untersucht wird.

5.1 Modellentwicklungszyklus

In dieser Lektion konzentrieren wir uns auf die Herausforderungen, die bei der Einführung eines Modells in die Produktion entstehen. Es bietet sich an, zunächst auf den Lebenszyklus der Modellentwicklung einzugehen. Beim Machine Learning geht es um die Erstellung eines Modells, das mit bestimmten Daten trainiert wird, um danach anhand von zusätzlichen Daten Vorhersagen treffen zu können. In diesem Kontext kann ein **Machine-Learning-Modell** oder ein statistisches Modell als Black Box betrachtet werden, das mithilfe eines bestimmten Inputs ein Output liefert. Die Grundidee besteht darin, dass anhand des Modells einem Input ein gewünschter Output zugeordnet werden kann. Beispielsweise könnte ein Computer so trainiert werden, dass er ein Bild einer Katze oder eines Hundes klassifizieren kann. Dabei werden dem Modell als Input Bilder und Bezeichnungen, die das Objekt des Bildes als Katze oder Hund deklarieren, geliefert. Die Black Box lernt die Beziehung zwischen den Bildern und den Bezeichnungen, um später die richtige Antwort (Hund oder Katze) in Hinblick auf neue Bilder geben zu können.

Machine-Learning-Modell

Eine Datei, die zur Erkennung von bestimmten Musterarten trainiert wurde.

Für die Entwicklung eines Modells, das einer Organisation von Nutzen ist, sollten...

1. ... das Problem verstanden werden,
2. gute Annahmen und Hypothesen aufgestellt werden,
3. verfügbare Daten gesammelt werden,
4. Untersuchungen zur Realitätstreue der Annahmen und Hypothesen gemacht werden,
5. mit möglichen Modellen und Datenverarbeitungsschritten zur Lösung des Problems experimentiert werden,
6. ein Modell trainiert und
7. in der Entwicklungspipeline bereitgestellt werden, sodass das richtige Verhalten getestet werden kann.

Bevor wir einsteigen, sollte erwähnt werden, dass alle Schritte miteinander verknüpft sind und nicht als Einzelschritte betrachtet werden können. So werden Daten während einer Datenbereinigung beispielsweise zu einem gewissen Grad auch analysiert, und nach einer Untersuchung muss der Datenbereinigungsprozess gelegentlich angepasst werden, da möglicherweise mit dem Störgeräusch zu viele andere Daten oder nicht genügend Daten entfernt wurden.

Problemverständnis und Hypothesen

Es ist wichtig, das Problem, um das es geht, genau zu verstehen. Bestenfalls ist der Projektrahmen genau definiert. Häufig sind Anforderungen jedoch eher vage formuliert, wie z. B. „Umsatz verbessern“, „Personalwechsel vermindern“ oder „zahlungsunwillige Kundenschaft identifizieren“. Die Suche nach Antworten auf die Frage, was für den Erfolg Ihres Projekts notwendig ist, wird als Anforderungsermittlung bezeichnet. Dazu müssen verschiedene Projektmanagementtechniken angewendet werden. Um das Problem zu verstehen und zu definieren, sind möglicherweise lange Besprechungen mit Fachkräften, „Design Thinking“ und exploratives Modellieren notwendig. Häufig müssen die anfänglichen Hypothesen und Annahmen einige Male überprüft und an den aktuellen Wissensstand angepasst werden. Das Fachgebiet Datenwissenschaften gilt als vergleichsweise junge Disziplin, und vielen Unternehmen fällt es schwer, datenorientiert zu denken und Data Science-Projekte erfolgreich umzusetzen. Daher ist es wichtig, genau zu verstehen, was gelöst werden muss und dass es klare Erfolgsindikatoren gibt. Das lässt sich durch gut definierte, spezifische Probleme, die beantwortet werden können, bewerkstelligen. So ließe sich das Ziel „Problemkundschaft ausmachen“ präziser formulieren: „die Zahl derjenigen, die nach zwei Wochen noch nicht bezahlt haben, soll um 20 Prozent reduziert werden.“ Selbst ein vermeintlich konkreteres Ziel wie „Ermittlung der Anzahl von Fahrzeugen, die in ein Gebäude fahren“ könnte folgendermaßen verbessert werden: „mindestens 95 % aller Fahrzeuge, die in ein Gebäude fahren, werden genau identifiziert“. Manche Probleme sind schwer definierbar. Ein Beispiel dafür stellt die maschinelle Übersetzung dar, denn wie lässt sich eine Übersetzung auf einer Skala von „gut“ bis „schlecht“ einordnen? Der Bewertung liegt eine gewisse Subjektivität zugrunde. Menschen erkennen zwar eine schlechte Übersetzung, dennoch ist es nicht einfach, Kriterien einer guten Übersetzung auszumachen. Sollte sie nahe am Originaltext sein? Sollte der Text das Wesentliche enthalten, aber ansonsten eher „frei“ übersetzt sein? Oder liegt die Lösung in der Mitte? Haben wir das Problem einmal definiert, können wir Hypothesen darüber aufstellen, wie es gelöst werden kann, was wir dazu brauchen, welche Annahmen dazu geprüft werden

müssen, welche Korrelationen zur Zuordnung von Input und Output benötigt werden, und schließlich kann mit der Untersuchung verschiedener Lösungsmöglichkeiten begonnen werden.

Datensammlung, -bereinigung und -aufbereitung

Wenn wir das Problem verstanden und Hypothesen zur Lösung aufgestellt haben, müssen entsprechende Daten gesammelt werden. In manchen Fällen bedeutet Datensammeln lediglich Zugriff auf eine Datenbank mit allen, für unseren Gebrauch gekennzeichneten Daten.

Oft ist es allerdings komplizierter und die Daten müssen anhand des so genannten ETL-Prozesses (**Extract, Transform, and Load**) extrahiert, transformiert und geladen werden. In anderen Fällen sind die Daten innerhalb des Unternehmens womöglich nicht verfügbar, aber können aus öffentlichen Quellen entnommen werden. Es gibt bereits einige frei verfügbare Datenbestände und es kommen täglich neue hinzu. Das Problem öffentlicher Daten ist, dass sie kaum einen Wettbewerbsvorteil bieten, da sie schließlich frei verfügbar sind und zudem unsere bestimmten Anforderungen selten genau erfüllen.

Eine weitere Möglichkeit ist es, Daten selbst zu sammeln. Das ist normalerweise zeit- und ressourcenaufwändig, aber dadurch werden genau die Daten (zumindest sehr viel bessere Daten) gesammelt, die gebraucht werden, um das Problem zu lösen. Es könnten zum Beispiel ...

- ... Bewegungsdaten von Passagieren gesammelt werden,
- Aufnahmen mit Kameras gemacht werden, die mit einem Produkt verschickt werden,
- Sensoren auf Geräten eingebaut werden, um ihr Verhalten zu beobachten, oder
- Bestehende Daten von Menschen gekennzeichnet werden, um sie zum überwachten Lernen zu verwenden.

Die Sammlung der notwendigen Daten ist nicht nur eine wissenschaftliche Disziplin, sondern auch eine Kunst, und überlappt sich oft mit der Experimentplanung. Eine weitere Möglichkeit ist das sogenannte Datenscraping, was oft einfacher ist: es beschreibt die Datengewinnung von Websites oder aus Quellen, die eigentlich nicht direkt von Computern verwendet werden, und die Transformation dieser Daten in ein computerlesbares Format. Daten können dabei von Internetseiten, PDFs oder gescannten Dokumenten stammen. **Scraping** ist zeitaufwändig, da für jede Quelle eine spezielle Software entwickelt werden muss, und zudem heikel, da es sehr eng mit der Quelle verbunden ist. Selbst kleinste Änderungen an der Quelle wirken sich auf den Prozess aus, der daraufhin angepasst werden muss. Außerdem kann die Extraktion von Daten rechtliche Problem mit sich bringen, wenn die Daten ursprünglich für einen anderen Zweck gesammelt wurden (Import.io, 2017). Daten, ganz gleich, ob sie extrahiert, gesammelt oder aus einer Datenbank oder einem Datenspeicher stammen, müssen bereinigt und für das gesetzte Ziel formatiert werden. Dabei handelt es sich um ein weiteres umfassendes Thema, das als Datenaufbereitung oder „Munging“ bezeichnet wird. Datenaufbereitung oder Munging ist die Transformation und Zuordnung von Daten von einem bestimmten Format in ein anderes, um Daten für ein Projekt wertvoller zu machen und anzupassen. Häufig werden dabei bestimmte Einheiten (Felder, Zeilen, Spalten, Datenwerte usw.) innerhalb eines Datensatz-

Scraping

Technik, bei der Daten mithilfe von Computerprogrammen aus menschenlesbarem Output aus anderen Programmen extrahiert werden.

zes transformiert. Auch Vorgänge wie Extraktion, Parsing, Zusammenfügen, Standardisierung, Aufbesserung, Bereinigung, Konsolidierung und Filtern, um ein bestimmtes Output zu erzielen, gehören dazu.

Explorative Datenanalyse

Wenn Zugriff auf die Daten besteht, kann damit begonnen werden, sie auf ihren Gehalt zu untersuchen. Dieser Schritt wird **explorative Datenanalyse (EDA)** genannt und bezieht sich darauf, anfängliche Untersuchungen der Daten durchzuführen, um anhand von zusammenfassenden Statistiken und graphischen Darstellungen Muster oder Anomalien zu entdecken sowie Hypothesen und Annahmen zu testen.

Standardtools für EDA helfen dabei, Wertarten, Zahlen einzelner Werte, Arten von Datenpunkten, von Datendurchschnitten und von Datenmitteln zusammenzufassen. Sie unterstützen auch die Erstellung von Visualisierungen zur intuitiven Ansicht der Daten und die Erkennung von extrahierbaren Mustern. Da unsere Wahrnehmung visuell veranlagt ist, lassen sich Muster auf einem Bild leichter erkennen als in Tausenden oder Millionen von Datenzeilen. EDA ist ein explorativer Prozess. Fachkräfte aus den Datenwissenschaften erkunden verschiedene Möglichkeiten und forschen weiter nach, wenn etwas interessant aussieht.

Explorative Datenanalyse (EDA)
Analyse von Datensätzen zur Zusammenfassung der Hauptcharakteristika, oft mithilfe von visuellen Methoden.

Dabei handelt es sich um einen intuitiven Prozess, der Erfahrung voraussetzt, daher können wir hier nur ein einfaches Beispiel liefern. Wir werden eine kurze Untersuchung des Iris-Datensatzes anstellen. Dieser kleine Datensatz enthält drei Klassen von jeweils 50 Exemplaren, bei dem sich jede Klasse auf eine Irisart bezieht, und wird oft als Beispiel verwendet (Fisher, 1936). Im Rahmen von EDA besteht einer der ersten Schritte meist darin, die Daten anzusehen, indem die ersten Reihen untersucht werden. Diesen Schritt nehmen wir in pandas in Python vor (pandas, o. D.). Der Iris-Datensatz wurde als .csv-Format heruntergeladen und auf dem lokalen Rechner, als Datei mit dem Namen `iris.data.csv`, gespeichert. Die Daten lassen sich tabellarisch folgendermaßen darstellen:

Code

```
import pandas as pd  
iris_df = pd.read_csv('iris.data.csv') iris_df.head() # first 5 rows
```

Abbildung 47: Die ersten Zeilen eines Datensatzes mit .head()

	Kelchblatt-längein cm	Kelchblatt-breitein cm	Blütenblatt-längein cm	Blütenblatt-breite in cm	Art
0	5.1	3.5	1.4	0.2	Iris setosa
1	4.9	3.0	1.4	0.2	Iris setosa
2	4.7	3.2	1.3	0.2	Iris setosa
3	4.6	3.1	1.5	0.2	Iris setosa
4	5.0	3.6	1.4	0.2	Iris setosa

Quelle: Pedori (2020).

Dieser Datensatz hat vier numerische Werte und eine Beschriftung, nämlich „Art.“ Die zusammenfassende Statistik für die numerischen Werte lässt sich wie folgt berechnen:

Abbildung 48: Übersicht über einen Datensatz mit .describe()

In [16]:	1	iris_df.describe()			
Out[16]:		Kelchblatt-länge in cm	Kelchblatt-breite in cm	Blütenblatt-länge in cm	Blütenblatt-breite in cm
	Anzahl	150.000000	150.000000	150.000000	150.000000
	Mittel	5.843333	3.054000	3.758667	1.198667
	Standard	0.828066	0.433594	1.764420	0.763161
	Minimum	4.300000	2.000000	1.000000	0.100000
	25%	5.100000	2.800000	1.600000	0.300000
	50%	5.800000	3.000000	4.350000	1.300000
	75%	6.400000	3.300000	5.100000	1.800000
	Maximum	7.900000	4.400000	6.900000	2.500000

Quelle: Pedori (2020).

Es gibt tatsächlich insgesamt 150 Werte und man kann sich ein Bild vom Durchschnitt und der Verteilung der Werte machen.

Experimentieren, Funktions- und Modellauswahl

Wenn die Daten in brauchbarer Form geladen sind und wir in etwa wissen, was sie aussagen, können wir beginnen, mit Modellen zu experimentieren. Es werden nach Anwendungsfall mehrere, verschiedene Modelle getestet, angefangen mit dem einfachsten Modell, wie der linearen oder logistischen Regression, hin zu komplexeren, wie neuronalen Netzen und XGBoost. Im Experimentierzyklus wird mit ein paar Modellen begonnen. Diese werden getestet, um einen Eindruck von ihrem Ergebnis zu bekommen. Teil des Prozesses ist es, mögliche **Funktionen auszuwählen**. Die vorhandenen Daten können Noise enthalten, der per Definition keine Bedeutung hat, und manche Informationen könnten aufgrund des bereits bekannten Fachwissens extrahiert werden. Handelt es sich zum Beispiel um Daten über Verkehr, ist der Wochentag ein wichtiger Faktor. Wenn die Daten jedoch nur den Monatstag enthalten, muss der Wochentag berechnet und als Funktion hinzufügen werden. Beim Training eines Modells kommt der so genannte „Fluch der Dimensionalität“ ins Spiel. Je mehr Dimensionen wir haben, desto schwerer ist es, für ein Modell zu konvergieren, daher empfiehlt es sich in manchen Fällen, eine **Dimensionalitätsreduktion** durchzuführen. Zu diesem Zeitpunkt werden normalerweise auch mögliche Modelle in Betracht gezogen. Die Hauptarten der Machine-Learning-Ansätze sind überwachtes und unüberwachtes Lernen sowie Verstärkungslernen.

- Überwachtes Lernen ordnet Eingabedaten bekannten Ausgabedaten zu. Beispiel: die Zuordnung von Katzen- und Hunde Bildern in Kategorien oder die Vorhersage des Aktienmarktes.
- Unüberwachtes Lernen erkundet Muster in den Daten. Beispiel: Clustering von Dokumenten nach Themen durch Suche nach Ähnlichkeiten in den Dokumenten, ohne dass zuvor Themen festgelegt wurden.
- Verstärkungslernen untersucht, wie Agenten mit der Umgebung umgehen, indem positive Situationen belohnt und negative bestraft werden. Beispiel: Lernen von Video- oder Brettspielen.

Die Auswahl des richtigen Machine-Learning-Modells für eine bestimmte Aufgabe ist nicht einfach. Es gibt viele Machine-Learning-Modelle, und mehrere Python-Softwarepakete bieten Zugriff auf einige bzw. viele davon an. Häufig wird zwischen traditionellen Machine-Learning-Algorithmen, meist auf der Basis von statistischen Methoden, und Deep-Learning-Algorithmen, auf der Basis von neuronalen Netzen, unterschieden. In vielen Fällen können durch **Ensemble-Modellierung**, bei denen Ergebnisse aus mehreren Modellen zur Verbesserung der Genauigkeit zusammengefasst werden, bessere Resultate erzielt werden. In manchen Fällen wird die erste Ebene des Machine Learning durch Deep Neural Networks ausgeführt und die Ergebnisse werden dann durch einfachere, herkömmliche Methoden aggregiert. Nach der Auswahl eines Versuchsmodells werden die Daten in **Trainings-, Test- und Validierungsdatensätze** aufgeteilt. Dazu werden häufig Techniken, wie die k-fache Kreuzvalidierung verwendet, um die Datenmenge für das Training und zur Validierung zu maximieren.

Funktionsauswahl

Auswahl eines Teils der relevanten Funktionen zur Verwendung im Modell.

Dimensionalitätsreduktion

Transformation von Daten aus einem hochdimensionalen Raum in einen niedrigdimensionalen, wobei einige der bedeutungsvollen Eigenschaften der ursprünglichen Daten erhalten bleiben.

Ensemble-Modellierung

Modellierung mit mehreren Modellen, um die Vorhersageleistung im Vergleich zur Leistung mit einzelnen Modellen zu verbessern.

Trainings-, Test- und Validierungsdatensätze

Der Trainingsdatensatz wird zur Anpassung des Modells am Anfang verwendet; mithilfe des Validierungsdatensatzes wird die Performanz des Modells in verschiedenen Stufen gemessen. Der Testdatensatz wird zu einer unverzerrten Leistungsbeurteilung zum Schluss eingesetzt.

Hyperparameter

Parameter, dessen Wert zur Steuerung des Lernprozesses verwendet wird.

Hyperparameteroptimierung

Jedes Machine-Learning-Modell verfügt über bestimmte Parameter, die während des Lernprozesses automatisch angeglichen und die nicht manuell verändert werden. Es gibt jedoch auch die so genannten „**Hyperparameter**“, die von Benutzer:innen des Algorithmus ausgewählt werden können. Beim Deep Neural Network zum Beispiel kann die Tiefe des Netzes festgelegt werden. Auch die Art und Weise, wie Funktionen, die ins Modell einfließen sollen, ausgewählt werden, kann als Hyperparameter des Machine-Learning-Experiments angesehen werden. Die Wahl der Hyperparameter ist keine leichte, da das Training oft viel Zeit in Anspruch nimmt, so dass nicht alle möglichen Kombinationen berechnet werden können. Erfreulicherweise gibt es Algorithmen und Bibliotheken, die die Hyperparameteroptimierung unterstützen. Diese Tools finden Hyperparameter, die für den jeweiligen Datensatz und die vorliegende Anwendung optimiert sind.

Training

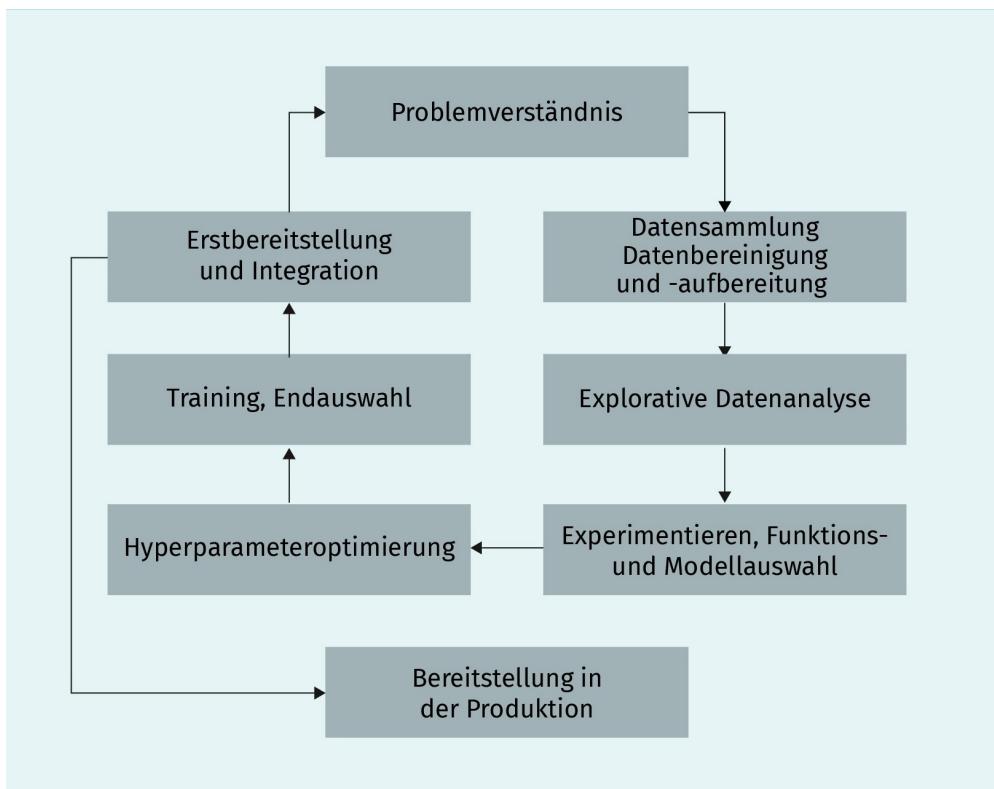
Nun kann das Training der verschiedenen Modelle beginnen, um ihre Performanz zu vergleichen. Es werden in der Regel mehrere Modelle mit demselben Datensatz auf verschiedene Funktionen getestet. Ist der Datensatz groß und das Training dauert lange, ist es oft nützlich, anfangs nur einen Teil des Datensatzes zu verwenden. Wenn Prozessvariablen, wie Vorbereitung, Modellierung und Funktionsauswahl, zuverlässig festgelegt werden konnten, kann das Modell noch einmal mit dem vollen Datensatz trainiert werden. Je nach Datenvolumen und Anwendungsfall kann das viel Zeit und viele Ressourcen in Anspruch nehmen. Ein extremes Beispiel dafür ist GPT-3 von OpenAI, das über 175 Milliarden Parameter verfügt und dessen Training auf aktueller Hardware 355 Jahre dauern und \$4.600.000 kosten würde (Brown et al., 2020). Unsere Fälle werden nicht so extrem sein, aber wir müssen dennoch ein vollständiges Training auf den Modellen, die sich anbieten, durchführen.

Das betrifft keine Modelle, die das Training in weniger als ein paar Stunden abschließen. Am Ende des Prozesses steht ein trainiertes Modell, dessen Performanz mit dem Testdatensatz gemessen werden kann.

Erstbereitstellung und Integration

Ist das Modell bereit, muss es auch verfügbar gemacht werden. Am einfachsten und am gebräuchlichsten ist es, es in einer REST-Schnittstelle als Microservice in einem Container bereitzustellen. Anschließend können einfache Usability-Tests stattfinden und das Modell kann mit anderen Diensten verknüpft und in einer Entwicklungspipeline bereitgestellt werden, um sicherzustellen, dass es wie erwartet funktioniert.

Abbildung 49: Modellentwicklung und Produktionslebenszyklus (I)



Quelle: Pedori (2020).

Unterschiede zur Softwareentwicklung

Beim Vergleich zwischen der traditionellen Softwareentwicklung und dem Machine Learning fällt auf, dass es in beiden Fällen unabdingbar ist, das Problem zu verstehen, um in der Lage zu sein, es zu lösen. In der traditionellen Softwareentwicklung sind Probleme häufig genauer definiert, zumindest in Hinblick auf die Anwendung. Softwareentwickler:innen verwenden gängige Softwarebausteine und können ihre Annahmen durch Iterationen prüfen.

Bei der traditionellen Softwareentwicklung spielen Hypothesen eine geringere Rolle als die Anforderungen. Es ist leichter, Verfahren für anforderungsbasierte Tests zu implementieren, da die Ergebnisse genauer sind. Im Gegensatz zu Machine-Learning-Modellen sollte der Code deterministisch sein, selbst wenn Interaktionen zu mehr Komplexität führen können.

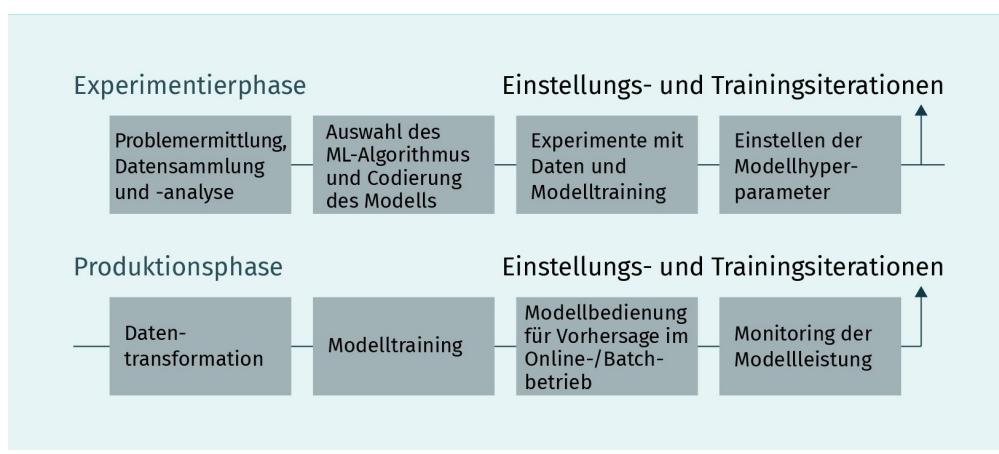
5.2 Modellproduktionszyklus

Software 2.0
Software 2.0 nutzt Machine Learning, genauer gesagt neuronale Netzwerke, wodurch die Art und Weise, wie Software geschrieben wird, fundamental verändert wurde.

Artefakt
Der Output des Trainingsprozesses.

Zum Zeitpunkt der Inbetriebnahme sind die Produktionsanforderungen von Machine-Learning-Modellen dem Produktionslebenszyklus von traditioneller Software sehr ähnlich. Es gibt jedoch auch bestimmte Unterschiede. Wie bei allen Codes sollte der Code auch hier in eine Pipeline für kontinuierliche Entwicklung und Bereitstellung integriert werden, so dass die schnelle Bearbeitung und häufige Lieferung einer zuverlässigen, sicheren und verfügbaren Software gewährleistet werden kann. Der Einsatz von Machine-Learning-Modellen zur Problemlösung wird **Software 2.0** genannt (Karpathy, 2017). Das gesamte ML-System enthält den Code zur Erstellung des Modells, das trainierte Modell selbst sowie möglicherweise andere **Artefakte** und die Daten, die zur Erstellung des Modells verwendet wurden. Dieser Code muss in die oben erwähnte Pipeline integriert werden. Um Modelle in der Produktion auf Dauer einzusetzen, muss Versionsverwaltung für den Code angelegt werden und der gesamte Code gespeichert und bereitgestellt werden. Zudem müssen die Endpunkte, in denen die Modelle angewendet werden, eingerichtet und betreut, für den Fortbestand der Vorhersagen gesorgt, die Ergebnisse protokolliert und Integrität und Leistung überwacht werden. Auch die Ergebnisse, die Modelle mit neuen oder nicht getesteten Daten produzieren, müssen überprüft werden.

Abbildung 50: Modellentwicklung und Produktionslebenszyklus (II)



Quelle: Kubeflow

Modelle in der Produktion

Sobald die Modelle die Produktionsphase erreichen, kommen zu den üblichen Schwierigkeiten von Softwareproduktion weitere spezielle Herausforderungen hinzu, da die Modelle selbst auf Software beruhen. Denken wir an einige der Dienste, die die meisten von uns tagtäglich nutzen.

Onlinesuchmaschinen sind Machine-Learning-Systeme. Die Suche in der Form eines Textstrings sollte eine Suchmaschine dazu anleiten, die Links zu den Webseiten, die Benutzende sehen wollen, anzuzeigen. Die Ergebnisse müssen genau, schnell und auf dem neuesten Stand sein, und das System sollte stets zur Verfügung stehen. Es wird ständig mit neuen Daten aus dem Netz versorgt, und es passt sich an die Benutzereinstellungen an.

Manche Streaming-Seiten arbeiten beispielsweise mit Empfehlungssystemen, die aggregieren, was andere Konsument:innen angesehen haben, kombiniert mit dem Verlauf eines:einer bestimmten Benutzer:in, um auf dieser Grundlage Filme vorzuschlagen. Das Ziel besteht darin, das Benutzer:inneninteresse aufrecht zu erhalten (und einige von uns können sicher bestätigen, dass dies funktioniert, wenn wir bedenken, dass wir eigentlich zu viel Zeit mit bestimmten Serien verbringen). Das System muss ständig verfügbar sein, extrem große Datenmengen liefern und sich dauernd an die sich ändernden Benutzer:innenvorlieben anpassen. Digitale Finanz- und Zahlungsdienste sind mit anderen Herausforderungen konfrontiert. Um zweckwidrige und betrügerische Aktionen schnellstmöglich und mit hoher Genauigkeit zu kennzeichnen, müssen Transaktionen kontinuierlich überwacht werden. Werden korrekte Transaktionen vom System blockiert und inkorrekte angenommen, so ist das für Benutzende äußerst lästig. Diese Systeme müssen ausfallsicher sein und ihre Leistung und Verzögerungen müssen stets überwacht werden.

Versionsverwaltung

Unter Versionsverwaltung wird die Verwaltung von Änderungen in Dokumenten, Computerprogrammen und anderen Informationssammlungen verstanden. Versionsverwaltung ist nötig zur Nachverfolgung von Änderungen, zur Wiederherstellung von funktionierenden Lösungen und zur Koordinierung von Updates. Zuvor wurde bereits **Git**, als ein Programm zur Versionsverwaltung, behandelt. Zahlreiche Tools für Machine-Learning-Verfahren verwenden Git oder beruhen darauf. In der Regel werden die Änderungen durch Nummern oder Buchstaben, den sogenannten Revisionen oder Revisionsnummern, markiert. Die erste Version eines Dateisatzes oder eines Codes könnte „Revision 1“ sein. Nach Änderungen wird sie zur „Revision 2“ usw. Anhand von Versionen und mithilfe eines **Versionsverwaltungssystems** können Änderungen (z. B. wer, was, wann geändert hat) nachverfolgt, verschiedene Versionen verglichen und Änderungen zwischen den verschiedenen Teilen einer Anwendung synchronisiert werden.

Versionsverwaltung von Code

Wenn ein Code schlicht als „Programmiersprache“ angesehen wird, verfügen ML-Systeme über zwei Codearten: den Implementierungscode und den Modellierungscode. Der Implementierungscode hält Code zum Zugriff auf APIs oder Systemintegrationscode, durch den ML-Systeme mit anderen Anwendungen verbunden werden, gewissermaßen zusammen. Modellierungscode wird zur Modellentwicklung verwendet. In manchen Fällen wird der Code in mehreren Sprachen geschrieben. Der Code muss zum Release versioniert und freigegebene Versionen müssen getestet werden. Zudem müssen vom Code abhängige Teile aktualisiert werden. Der neue Code hängt mit anderen Codeteilen zusammen. Bisher ist das genau der Vorgang, der auch im Entwicklungsprozess von herkömmlicher Software abläuft. Um das Problem des Umgebungsdrifts in der Releasepipeline zu beheben, wurde Infrastructure as Code (IaC) entwickelt. Wie bereits angemerkt, ist das ML-System jedoch nicht nur der Code zur Generierung des Modells, die Verpackung des Modells und die Verbindung zu anderen Diensten: Es enthält auch die Daten zur Generierung des Modells und die Modelle selbst, deren Versionen verwaltet werden müssen.

Git

GitHub ist eine häufig verwendete, auf Git basierende Plattform, auf der viele Open-Source-Projekte gehostet werden.

Versionsverwaltungssystem

Ein System, in dem Änderungen (z. B. wer, was und wann geändert hat) nachverfolgt und verschiedene Versionen verglichen werden können.

Versionsverwaltung von Daten

Metadaten

Daten über die Art der Spalten (oder einer Funktion) im Datensatz.

Die Daten zum Modelltraining können sich ändern. Die **Metadaten** (Format, Bezeichnungen, Reihenfolge und Anzahl der Spalten) können sich ändern, Werte können entfallen oder hinzugefügt werden, die Genauigkeit einiger Werte kann sich ändern oder die statistischen Eigenschaften im Vergleich zu neuen Daten können sich stärker ändern als die zur Erstellung des Modells verwendeten Daten. Um dasselbe Modell immer wieder generieren zu können – eine Anforderung zur zuverlässigen Bereitstellung – muss gewährleistet werden, dass dieselben Daten verwendet werden, die bereits bei der Lieferung genutzt wurden. Aus diesem Grund ist die Versionsverwaltung der Daten notwendig. Dies betrifft, wie bereits gesagt, sowohl die Daten als auch die Metadaten. Das Hauptproblem dabei ist die Datenmenge. Das Volumen von Trainingsdaten kann zwischen einigen Megabytes oder mehreren Petabytes variieren. Einige Megabytes können in einem normalen Versionsverwaltungssystem leicht gespeichert werden. Das Speichern von Gigabytes oder mehr ist jedoch schwieriger. Daher wurden Systeme entwickelt, in denen große Dateien gespeichert werden können: Angefangen mit Open Source-Add-Ons über Git (z. B. Git Large File Storage (LFS) (GitHub, o. D.-c) und Data Version Control (DVC) (DVC, o. D.)) bis hin zu einigen proprietären Lösungen. Ein großes Problem der Datenversionsverwaltung stellen die Speicheranforderungen von potenziell großen Datenmengen dar. Des Weiteren müssen die Metadaten die Modellanforderungen auch auf Dauer erfüllen.

Versionsverwaltung von Modellen

Die Versionen von Modellen und anderen Artefakten, die während der Modellentwicklung generiert wurden, müssen verwaltet werden, so dass sie dem Algorithmus, mit dem sie generiert wurden, dem Code, der sie unterstützt und der sie verwendet, und den Trainingsdaten angepasst werden können. Die Speicheranforderungen von Modellen sind weniger problematisch als die für die Daten, selbst wenn Modelle Hunderte von Megabytes benötigen und ein Problem für normale Versionsverwaltungssysteme darstellen. Die maximale Dateigröße von Git ist 100mb. Für alles, was darüber hinausgeht, kann das bereits erwähnte Git Large File Storage in Erwägung gezogen werden (GitHub, o. D.-c).

Reproduzierbares Training von Modellen

Ein bestimmtes Modell sollte bei Bedarf aus einer bestimmten Datenversion und dem zur Modellerstellung verwendeten Code neu erstellt werden können. Da dies jedoch Zeit und Ressourcen in Anspruch nimmt, muss die Modellversion mit allem, was dazu gehört, gespeichert werden. Durch Versionsverwaltung wird gewährleistet, dass zu verschiedenen Zeitpunkten und auf verschiedenen Maschinen aufgrund derselben Daten- und Codeversionen dieselben Resultate erzielt werden können.

Bereitstellung

Nach Entwicklung und Training eines Modells muss es zur Verwendung in einer Organisation bereitgestellt werden. Dazu gibt es zwei Möglichkeiten:

- **Eingebettetes Modell:** Das Modell als Artefakt (die tatsächliche Datei) wird kompiliert und als Paket mit der Anwendung, die es verwendet, gebündelt. Ab diesem Zeitpunkt ist es Teil der Anwendungsressourcen, genau wie das Hintergrundbild einer Website.
- **Modell als Service:** Das Modell wird von einem Dienst umschlossen und kann so, unabhängig von der Anwendung, in der es verwendet wird, bereitgestellt werden. Dies ermöglicht eine Entkopplung, kann jedoch die Latenz erhöhen, und die Anwendungen müssen eine Art entfernten Anruf ausführen.

In beiden Fällen müssen die Modelle gespeichert und nach Erstellung an die entsprechende Infrastruktur geliefert werden. Eine Bereitstellung als Dienst ist komplexer, aber ein Dienst kann dabei als Anwendung verstanden werden: Dabei wird das Modellartefakt gespeichert und in die Anwendung eingebettet, für die es erstellt wurde. Die wesentlichen Kriterien für die Bereitstellung stimmen weitgehend mit denen der traditionellen CI/CD-Pipelines für die Softwareentwicklung überein, bis hin zu den Integrations- und Systemtests. Dennoch können sich bei der Modellbereitstellung andere komplexe Szenarien ergeben.

Mehrere Modelle

Manchmal kommt es vor, dass für eine Aufgabe unterschiedliche Modelle zur Verfügung stehen oder mehrere Modelle für die Arbeit an einer Aufgabe zum Einsatz kommen. Wir können diese Modelle entweder als eigenständige Dienste bereitstellen oder mit einem einzigen API-Aufgriff auf mehrere Modelle zugreifen. Ersteres bietet mehr Flexibilität, allerdings müssen die verwendeten Anwendungen entsprechend erweitert werden. Das zweite Szenario ermöglicht es, die Anzahl und Art der Modelle zu ändern, ohne dass die restliche Anwendung umgeschrieben werden muss.

Schattenmodelle

In manchen speziellen Fällen lohnt es sich, mehrere Versionen des gleichen Modells nebeneinander in der Produktionsumgebung zu haben. Damit lässt sich überprüfen, dass die neuen Modelle, von denen erwartet wird, dass sie besser als die bisherigen funktionieren, das tatsächlich tun. Zu diesem Zweck sammeln wir Daten über das Schattenmodell, bevor es in die aktive Produktion integriert wird. Diese Vorgehensweise ähnelt dem Testen, findet jedoch mit tatsächlichen Daten statt: Somit kann sichergestellt werden, dass sich das neue Modell wie erwartet verhält, bevor wir es in Betrieb nehmen.

Konkurrierende Modelle

Bei komplexeren Szenarien können wir mehrere Versionen desselben Modells in der Produktion behalten, sodass, ähnlich wie in einem **A/B-Test**, geprüft werden kann, welches am besten ist. Dadurch wird das Projekt in Hinblick auf Infrastruktur und Routing komplexer, da sichergestellt werden muss, dass die Daten bei den richtigen Modellen ankommen und dass genügend Daten vorhanden sind. Nur so können wir entscheiden, welches Modell weitergeführt werden soll.

A/B-Test

Test aus einem randomisierten Experiment mit zwei Varianten, A und B.

Online-Learning-Modelle

Die bisher behandelten Modelle werden entwickelt, trainiert und dann bereitgestellt. Onlinemodelle werden ständig aktualisiert und können ihre Leistung kontinuierlich anhand neuer Daten verbessern, d. h. sie lernen, während sie produktiv sind. Auch dadurch wird das Projekt komplexer, denn die Modelle sind keine statischen Artefakte und ihre Ergebnisse verändern sich stets. Es muss gewährleistet werden, dass die Modellleistung nicht nachlässt, dass die zum Training verwendeten Daten geeignet sind und dass die Versionen der aktualisierten Modelle und Produktionsdaten verwaltet werden, sodass auf eine gut funktionierende Version zurückgegriffen werden kann, falls ein Problem auftritt. Dieses Szenario ist recht komplex und kommt außerhalb von hochspezialisierten Anwendungen in sehr großen Organisationen selten vor.

Zur Unterstützung von komplexeren Bereitstellungsszenarien empfiehlt sich die Verwendung von flexibler Infrastruktur, die bei Bedarf angepasst werden kann. Dazu sind Cloudanbieter notwendig, auf die später eingegangen werden soll.

Testverfahren und Qualität

In der Softwareentwicklung kann mithilfe von Tests sichergestellt werden, dass ein System funktioniert. Darüber hinaus zeigen uns Tests, wenn etwas nicht korrekt abläuft. In ML-Umgebungen ist das Testen noch umfangreicher als bei der Entwicklung von herkömmlicher Software, da Daten, Training und die Integration mit anderen Komponenten getestet werden müssen.

- **Daten:** Es können Tests zur Validierung von Eingabedaten gegen das erwartete Schema oder von Annahmen zu gültigen Werten vorgenommen werden. Es muss gewährleistet werden, dass die Daten, die ins Modell eingeführt werden, den Daten, die fürs Training des Modells verwendet wurden, entsprechen. Dies ist sogar noch wichtiger, wenn zu erwarten ist, dass das Modell ständig mit Betriebsdaten aktualisiert wird.
- **Komponentenintegration:** Das Zusammenspiel, Verhalten und die anhand von festgelegten Protokollen und Verträgen definierte Kommunikation von verschiedenen Komponenten eines Systems sollten gut funktionieren. Zudem muss sichergestellt werden, dass sich Komponenten in der Produktionsumgebung genauso verhalten wie in der Entwicklungsumgebung, d. h. die Validierung desselben Datensatzes sollte zu denselben Ergebnissen führen. Das klingt einfacher, als es ist und jegliche Unzulänglichkeiten sollten untersucht werden.
- **Modellqualität:** Die Leistung eines ML-Modells ist nicht-deterministisch, aber relevante Werte sollten zur Beurteilung der Modellleistung gesammelt und überwacht werden. Dieses Thema wird im Folgenden noch weiter untersucht. Vor allem sollten sich die maßgeblichen Werte innerhalb eines annehmbaren Bereiches bewegen, besonders wenn das Modell neu trainiert werden muss.
- **Verzerrung und Fairness:** Das Modell sollte Ergebnisse erzielen, die mit den Organisationszielen im Einklang sind. Dieses Thema geht jedoch über den Rahmen dieses Kurses hinaus.

Monitoring und Einblick ins System

Nach Beginn des Produktivbetriebs muss die Leistung des Modells in der Produktion untersucht und die Feedbackschleife zum Entwicklungsprozess geschlossen werden. Zu diesem Zeitpunkt lassen sich zuverlässige Daten über das bisher Entwickelte sammeln. Die Haupttools sind Standardtools zur Überwachung von Software im Produktionsbetrieb mit besonderer Anwendung auf den ML-Prozess. Bei Software im Produktionsbetrieb werden Tools zur Aggregation von Protokollen, zur Kennzahlensammlung und zur Erfassung von Daten aus dem Produktivsystem verwendet: Leistungskennzahlen, Softwarezuverlässigkeit, Leistung, Informationen zum Debuggen, falls Fehler auftreten, und andere Indikatoren darüber, dass unerwartete Vorgänge aufgetreten sind. Im Allgemeinen sollte eine Benachrichtigung eingehen, wenn etwas falsch läuft oder wenn etwas Unerwartetes passiert, so dass Untersuchungen angestellt werden können. Bei Machine-Learning-Systemen im Produktionsbetrieb sollten besonders folgende Aspekte nachverfolgt werden:

- **Modellinput:** Daten, die ins Modell einfließen, anhand derer Verzerrungen, die bei den vorgesetzten Schritten auftreten könnten, gemessen werden können.
- **Modelloutput:** Vorhersagen, Empfehlungen und Ergebnisse, die das Modell auf die Eingabedaten hin produziert, um das Verhalten des Modells mit tatsächlichen Daten zu verstehen. Dazu gehören unter Umständen auch die Entscheidungen, die aufgrund der Outputs getroffen werden, da manchmal, wenn das System die Modelloutputs in Sonderfällen verwirft, Ausnahmen definiert werden müssen.
- **Output zur Interpretation des Modells:** Anhand von diesen Messwerten können die Vorhersagen des Modells weiter untersucht werden, um mögliche Überanpassung oder Verzerrungen, die während des Trainings nicht entdeckt wurden, zu ermitteln. Amazon hatte beispielsweise ein Machine Learning-Tool zur Vorauswahl von Lebensläufen eingeführt. Zunächst schien es sehr gut zu funktionieren, bis entdeckt wurde, dass ein Bias gegen Frauen bestand, da der Trainingsdatensatz nur wenige Anstellungen von Frauen enthielt. Heute wird dieses Tool nicht mehr von Amazon eingesetzt (Dastin, 2018).
- **Benutzer:innenaktion und Anerkennung:** Das Benutzer:innenverhalten nach Erhalt des Outputs sollte verfolgt werden. Kaufen Interessierte das, was ihnen präsentiert wurde? Schauen sie sich den vorgeschlagenen Film an? Zählen sie ihren Kredit termingerecht ab? Diese Daten sind enorm wichtig, wenn geprüft wird, ob das richtige Problem gelöst wurde.
- **Fairness des Modells:** Wie bereits erwähnt, sollte das Modellverhalten im Hinblick auf Rasse, Geschlecht, Alter usw. beurteilt werden. Dabei sollte auch geprüft werden, ob das Modellverhalten den Werten einer Organisation entspricht.
- **Rechenleistung des Modells:** Wie bei allen Softwaresystemen kann die Reaktionszeit entscheidend sein. Bei manchen Benutzer:inneninteraktionen ist bereits eine Verzögerung um wenige Sekunden genauso wertlos wie gar keine Reaktion. Die Reaktionszeiten des Modells, des CPU und die Arbeitsspeicherauslastung sollten nachgehalten werden.

Sind mehrere Modelle im Produktionsbetrieb, ist es umso wichtiger, Daten zu sammeln, zu überwachen und zu beobachten. Zur Beurteilung eines Schattenmodells kann ein A/B-Test ausgeführt oder verschiedene Experimente durchgeführt werden. Die gesammelten Daten sind zur Schließung der Datenfeedbackschleife unerlässlich: Anhand von tatsächlichen Daten oder **mit menschlicher Hilfe** bei der Analyse neuer Daten aus der Produktion können neue Datensätze gewonnen werden, die zur Erstellung neuer, idealerweise verbesserte-

Human in the Loop (HITL)

Modell mit menschlichem Einsatz im Gegensatz zum geschlossenen Modell, das ohne menschliche Interaktion auskommt.

Model Drift

Leistungsminderung eines Modells aufgrund von Änderungen der Eingabedaten oder der Trainingsdaten.

ter Modelle führen. So können wir lernen, unsere Modelle auf Grundlage ihres Verhaltens und anhand von realen Produktionsdaten anzupassen, was eine kontinuierliche Verbesserung ermöglicht.

Model Drift

Daten können sich im Lauf der Zeit verändern, sodass die Leistung der Modelle nachlässt, wenn diese sich nicht den Daten anpassen. Dieses Problem wird als „**Model Drift**“ bezeichnet (Brownlee, 2019).

Es gibt zwei grundlegende Arten des Model Drifts:

- **Concept Drift:** Diese liegt vor, wenn sich die statistischen Eigenschaften des gewünschten Outputs ändern. Beispielsweise kann die Vorhersage des Ausgabeverhaltens von Kunden nicht genau erfolgen, wenn das Modell die Verkaufsangebote nicht berücksichtigt (Wikipedia, 2020d).
- **Data Drift:** Diese liegt wiederum vor, wenn sich die statistischen Eigenschaften des Inputs im Vergleich zu den im Training verwendeten Daten ändern. Beispielsweise kann ein Modell zum Verkauf von Getränken, das im Sommer trainiert wurde, im Winter ohne Anpassungen nicht funktionieren.

Änderungen an den Daten können verschiedene Formen annehmen. Konzeptuell kann dies wie folgt modelliert werden:

- **Allmähliche Veränderung im Lauf der Zeit:** Manche Trends beeinflussen Daten, z. B. wenn Menschen aus der Stadt aufs Land ziehen.
- **Wiederkehrende oder zyklische Änderungen:** Ein typisches Beispiel hierfür stellen Jahreszeiten oder Wochentage dar, genau wie Feiertage, Quartalsende usw.
- **Plötzliche oder abrupte Änderungen:** Die Coronavirus-Pandemie wirkte sich beispielsweise auf Flüge und die gesamte Wirtschaft aus. Die Einführung des iPhones wirkte sich auf die Verkaufszahlen anderer Smartphones aus.

Um mit Model Drift umzugehen, können verschiedene Aspekte berücksichtigt werden (Zlobaite, 2010):

- **Zukünftige Annahmen:** Annahmen über zukünftige Datenquellen; wie diese sich von bestehenden unterscheiden; wie Daten überhaupt erlangt werden können; und wie diese ins System integriert werden können.
- **Änderungsart:** Mögliche Änderungsmuster erkennen; welche Änderungen vorhergesagt werden können; und wie Anpassung an sie erfolgen soll. **Saisonale Schwankungen** können leichter bedacht werden, Trends sind am schwersten zu erkennen.
- **Selbstadaptivität des Systems:** Festlegen, wie das Modell hinsichtlich der Änderungsart und der zukünftigen Annahmen angepasst werden soll.
- **Modellauswahl:** Ein Kriterium zur Auswahl einer bestimmten Parametrierung des ausgewählten lernenden Systems in verschiedenen Intervallen.

Es bestehen unter anderem folgende Möglichkeiten, um auf Model Drift zu reagieren:

Saisonale Schwankungen

Schwankungen, die in bestimmten, regelmäßigen Intervallen auftreten, die kürzer als ein Jahr sind, also wöchentlich, monatlich oder vierteljährlich.

- **Keine Reaktion:** Dabei wird angenommen, dass die Daten sich nicht ändern und ein statisches Modell daher akzeptabel ist. In manchen Fällen ist es am besten, nichts zu unternehmen.
- **Erneutes Training in regelmäßigen Abständen:** Statische Modelle können immer wieder durch die neuesten historischen Daten aktualisiert werden. Ein Modell kann z. B. jeden Monat aktualisiert und das neue Modell daraufhin erneut als statisches Modell eingesetzt werden. In manchen Fällen liegt nur eine geringe Auswahl neuer Daten oder ein „Sliding Window“ vor.
- **Datengewichtung:** Einige Algorithmen ermöglichen die Gewichtung von Eingabedaten. Beispielsweise können jüngere Daten stärker gewichtet werden.
- **Auswahl eines Modells:** Manchmal können Systeme entwickelt werden, welche Änderungen erkennen und ein bestimmtes anderes System zur Vorhersage auswählen können. Eine derartige Kombination eignet sich dann, wenn schnell auf plötzliche Änderungen und ohne menschliches Zutun reagiert werden muss, wie z. B. an der Börse, wo ein Algorithmus schnell auf eine Krise reagieren können muss, ohne dass er dazu umgeschrieben werden muss.
- **Datenaufbereitung:** Bei Zeitreihenproblemen wird davon ausgegangen, dass sich die Daten im Lauf der Zeit ändern. Daher können systematische Änderungen zum Teil von vornherein aus dem System gezogen werden. Das funktioniert allerdings nur, wenn die Änderungen modellierbar sind. Streng genommen handelt es sich bei systematischen Änderungen eigentlich nicht um Modell Drift, da sie von vornherein berücksichtigt werden können. Manche Änderungen sind jedoch nicht systematisch, und das führt zu besonderen Herausforderungen für Modelle in der Finanzbranche, aber auch in anderen Branchen.

Probleme und Problembehebung

Nachdem wir die Schritte, die bei der Inbetriebnahme eines Modells zu beachten sind, und die potenziellen Schwierigkeiten skizziert haben, wollen wir nun auf den Zweck hinter den Schritten eingehen: Deren Hauptziel besteht darin, Probleme zu minimieren und den Wiederherstellungsprozess, nachdem ein Problem aufgetreten ist, so einfach wie möglich zu gestalten. In der Produktion wird früher oder später etwas schieflaufen. In dem Fall müssen sowohl die Schäden minimiert als auch die Wiederherstellung gewährleistet werden. Welche Probleme können auftreten und wie können die oben genannten Aspekte für Abhilfe sorgen? Dauert die Produktion zu lang, muss ein Modell aufgrund von Modell Drift, neuen Herausforderungen, veränderten Geschäftsanforderungen, neuen Daten oder anderen unerwarteten Ereignissen erneut trainiert werden. Dazu ist Versionsverwaltung nötig, und es muss gewährleistet werden, dass die Version auch die zum Training des Modells verwendeten Daten enthält. Ausfälle oder Cyberangriffe können dazu führen, dass die Infrastruktur zum Teil unbrauchbar wird. Die Modelle müssen jedoch trotzdem wiederhergestellt werden und wie geplant funktionieren. Um zu entscheiden, wann die oben behandelten Maßnahmen eingesetzt werden sollen, sind Daten aus dem Monitoring notwendig. Diese werden auch benötigt, wenn mehr Ressourcen gefordert sind, da die Leistung des Systems nicht ausreicht. Nach der Aktualisierung eines Modells muss geprüft werden, ob das neue Modell wie erwartet funktioniert (auch hierzu ist Monitoring notwendig), und wenn das nicht der Fall ist, sollte ein Rollback auf eine ältere Version möglich sein (dazu ist Versionsverwaltung nötig). Wenn ein Modell ständig mit aktuellen Daten ver-

sorgt und damit trainiert wird, muss sichergestellt werden, dass die Leistung nicht abfällt. Falls das passiert, muss ein Rollback möglich sein, und auch hier sind Monitoring, Versionsverwaltung und Reproduzierbarkeit erforderlich.

5.3 MLOps und DataOps

Wie in vorherigen Abschnitten erläutert, sind mit der Inbetriebnahme und Instandhaltung eines ML-Systems, einschließlich des Modells, einige Schwierigkeiten verbunden. Diese Vorgänge in den Datenwissenschaften entsprechen den DevOps der traditionellen Softwareentwicklung. DevOps kommt in der Regel bei der Entwicklung und im Betrieb von großen Softwaresystemen zur Anwendung. Es bringt Vorteile mit sich, wie z. B. kürzere Entwicklungszyklen, schnellere Bereitstellung und zuverlässige Freigaben. Die beiden wichtigsten Vorgänge dafür stellen die kontinuierliche Integration und die kontinuierliche Lieferung dar. Ein Machine-Learning-System ist ein Softwaresystem. Das heißt, dass wir diese Techniken mithilfe der folgenden Kriterien entsprechend anpassen können.

Teamfähigkeiten

Datenwissenschaftler:innen sind Teil eines Teams, sie sind aber nicht immer Informatikfachleute. Der von ihnen geschriebene Code muss oft nachgearbeitet werden, um Produktionsqualität zu gewährleisten.

Entwicklung

Machine Learning kann als experimentell und nichtdeterministisch beschrieben werden. Daher muss mit verschiedenen Funktionen, Algorithmen, Konfigurationen, Datensätzen und Problembeschreibungen experimentiert werden, um so schnell wie möglich zu einer Lösung zu gelangen. Es ist nicht einfach, festzuhalten, was funktioniert und was nicht zum Erfolg geführt hat und gleichzeitig die Reproduzierbarkeit und die Wiederverwendbarkeit des Codes zu gewährleisten.

Testen

Wie bereits erwähnt, kommen beim Testen Elemente zum Tragen, die bei anderen Softwaresystemen nicht auftreten. Neben den typischen Modul- und Integrationstests muss für die Qualitätsbeurteilung des trainierten Modells sowie für Daten- und Modellvalidierung gesorgt werden.

Bereitstellung

Die Bereitstellung von Machine-Learning-Systemen muss oft in einer mehrstufigen Pipeline stattfinden, sodass Modelle automatisch neu trainiert und bereitgestellt werden können. Das erhöht die Komplexität und die Notwendigkeit, Schritte zu automatisieren, die vor der Bereitstellung manuell von Fachleuten ausgeführt wurden.

Produktion

Die Leistung von Machine-Learning-Systemen kann durch die Qualität des Codes und durch Änderungen des Datenprofils beeinträchtigt werden. Modelle können im Vergleich zu anderen Softwaresystemen in mehrfacher Hinsicht an Qualität verlieren. Dieser Verfall muss mithilfe von Messwerten, Monitoring, automatischen Benachrichtigungen oder automatischem Rollback auf funktionierende Versionen beobachtet bzw. behoben werden.

Es werden also sowohl für Machine-Learning-Systeme als auch für andere Softwaresysteme die kontinuierliche Integration von Quellcodeverwaltung, Modul- und Integrations- tests und die kontinuierliche Lieferung des Softwaremoduls oder des Pakets benötigt. Es gibt jedoch auch einige Unterschiede. Bei der kontinuierlichen Integration geht es nicht mehr ausschließlich um das Testen und Validieren von Code und Komponenten, sondern auch um das Testen und Validieren von Daten, Datenschemata und Modellen. Genauso dreht sich die kontinuierliche Lieferung nicht nur um ein einzelnes Softwarepaket oder einen Dienst, sondern um ein System, d. h. eine ML-Trainingspipeline, in der automatisch auch ein anderer Dienst (nämlich der Modellvorhersagedienst) bereitgestellt werden sollte. Hinzu kommt der Aspekt des kontinuierlichen Trainings. Dieses ist eine neue Eigenschaft, die nur auf ML-Systeme zutrifft und bei der es um das automatische Training und die Unterstützung der Modelle geht (Google, 2020a). Diese unterschiedlichen bzw. neuen Anforderungen führen zu zwei zusätzlichen Praktiken für datenintensive Vorgänge, nämlich DataOps und MLOps. DataOps (aus dem Englischen von „data analytics“ für Datenanalyse und „operations“ für Betrieb) ist der Einsatz von automatischen, prozessorientierten Methoden in Datenteams zur Qualitätsverbesserung und Zyklusverkürzung der Datenanalyse im gesamten Lebenszyklus der Daten, von der Datenaufbereitung bis zur Berichterstellung. Der leitende Analyst von Blue Hill Research, Toph Whitmore, drückte dies wie folgt aus (Vorhies, 2017):

1. Fortschritt und Leistung sollten bei jedem Schritt im Datenfluss gemessen werden.
2. Daten und Metadaten müssen mithilfe von Regeln definiert werden.
3. Personen in der Feedbackschleife sollten prüfen, ob die Annahmen korrekt sind.
4. Es sollten so viele Phasen wie möglich automatisiert werden.
5. Engpässe müssen aufgezeigt werden.
6. Governance Compliance, d. h. Datenkontrolle, Besitz der Daten, Transparenz und Nachverfolgung, sollte in den Prozess integriert sein.
7. Beim Design müssen Wachstum und Erweiterbarkeit, z. B. zunehmendes Datenvolumen und -vielfalt, berücksichtigt werden.

MLOps (von „Machine Learning“ und „operations“ (Betrieb)) ist die Praktik der Zusammenarbeit und Kommunikation von Datenwissenschaftler:innen und Betriebsfachleuten zum gemeinsamen Management des Produktionslebenszyklus von ML-Systemen, wie im vorherigen Abschnitt beschrieben. Dazu gehören:

- Bereitstellung und Automation,
- Reproduzierbarkeit von Modellen und Vorhersagen,
- Diagnoseverfahren,
- Governance und Einhaltung gesetzlicher Bestimmungen,

- Skalierbarkeit,
- Zusammenarbeit,
- Geschäftsanwendungen,
- Monitoring und Management.

Andere Bezeichnungen und kursierende Begriffe sind AIOps für KI-gestützte Betriebsabläufe, ModelOps für die Automatisierung von ML-Modellen und MLOps speziell für Deep Learning. Dabei handelt es sich um Unterkategorien von MLOps. Da das Interesse an und die Verwendung von Machine-Learning-Systemen in den letzten Jahren explosionsartig angestiegen ist, wurden mehrere Tools zur Unterstützung von MLOps entwickelt: **Airflow**, Luigi, Argo, Kubeflow, MLFlow, Michelangelo und viele andere. Zum Teil verfügen sie über die gleichen Funktionen, zum Teil lösen sie verschiedene Probleme auf unterschiedliche Art und zum Teil schreiben sie Lösungsansätze vor, die entweder gut oder weniger gut mit dem Workflow einer Organisation harmonieren. Im Detail werden MLFlow zur Modellverfolgung, Kubeflow für ML-Pipelines und Michelangelo für den gesamten ML-Prozess von Anfang bis Ende behandelt. Diese Lösungen lassen sich in die bestehende Infrastruktur einer Organisation integrieren. In manchen Fällen sind dazu nur kleine Änderungen notwendig, bei anderen müssen Anpassungen am im Tool impliziten Modell vorgenommen werden.

Airflow

Apache Airflow ist eine Workflowmanagementplattform und quelloffen erhältlich.

Modellverfolgung mit MLFlow

Mit MLFlow kann die Modellentwicklung und -verfolgung so automatisiert werden, dass ein optimales Modell gewählt werden kann (MLFlow, o. D.). Parameter, Attribute und Leistungszahlen lassen sich damit festhalten, und die Modelle, die bestimmten Kriterien entsprechen, können gekennzeichnet werden. Die Hauptfunktion von MLFlow ist unter anderem die Verfolgung der ML-Experimente, indem Parameter, Ergebnisse, Modelle und Daten für jeden Versuch nachverfolgt werden. In Bezug auf Machine-Learning-Bibliotheken, Algorithmen, Bereitstellungstools oder Sprachen ist MLFlow agnostisch.

Es ist so konzipiert, dass es leicht zu bestehendem Machine-Learning-Code hinzugefügt werden und der Code innerhalb einer Organisation anhand der Machine-Learning-Bibliothek geteilt werden kann, so dass auch andere damit arbeiten können. Obwohl sich MLFlow hauptsächlich für die Modellwahl-Phase im Lebenszyklus eignet, kann es in Verbindung mit anderen Tools, wie Airflow, den gesamten ML-Lebenszyklus unterstützen. MLFlow hat ein modulares und API-basiertes Design, wobei die Funktionalität in vier Teile geteilt ist: Verfolgung, Projekte, Modelle und Registrierung. Die Verfolgung ist das Zentrum, wo Details zu den Modellen wie in einem Metaspeicher abgefragt werden können. Der Client kommuniziert mit dem Protokollierungsserver anhand des HTTP-Protokolls und mit APIs in Python, REST, R und Java. Der Protokollierungsserver erfasst Detailinformationen zu den Modellen und verwendet Backendspeicher, um Folgendes abzulegen:

- Protokollierungsparameter,
- Codeversionen,
- Messwerte,
- Artefakte (Modelle und Datendateien),
- Start- und Endzeit der Ausführung,
- Tags und Notizen.

MLFlow Projects

MLFlow Projects enthält organisierten und verpackten Code, mit dem die Reproduzierbarkeit eines Models gewährleistet werden kann. Dazu wird die in YAML geschriebene MLProject-Datei verwendet. Sie beschreibt die Anforderungen des Machine-Learning-Projekts.

Ein einfaches Beispiel für ein MLProject:

Code

```
name: sklearn-demo
conda_env: conda.yaml
entry_points:
    model_run:
        parameters:
            max_depth: int
            max_leaf_nodes: {type: int, default: 32}
            model_name: {type: string, default: "tree-classification"}
            run_origin: {type: string, default: "default" }
        command: "python model_run.py -r {max_depth}{max_leaf_nodes} {model_name}"
```

In diesem Beispiel wird ein Entscheidungsbaum definiert, indem ...

1. ... der Name `sklearn-demo` festgelegt wird.
2. Die **Anaconda**-Umgebung `conda.yaml` deklariert wird.
3. Folgende Parameter festgelegt werden: `max_depth` - als Pflichtwert für die maximale Tiefe des Baums als Integer; `max_leaf_nodes` - ein Integer mit 32 als Standardwert, die maximale Zahl von Blattknoten des Baums; `model_name` - der Name des Models als String; `run_origin` - der Beginn der Ausführung.
4. Der Befehl, der ausgeführt werden soll, festgelegt wird, nämlich der Python-Interpreter auf die Datei `model_run.py`.

Anaconda

Eine Distribution der Programmiersprachen Python und R für wissenschaftliches Rechnen.

MLFlow Models

Mit MLFlow Models lassen sich Formate für die Verpackung von ML-Modellen definieren, die in verschiedenen nachgeschalteten Tools, wie REST-API oder Apache Spark, verwendet werden können. Das Format ist so angelegt, dass ein Modell in verschiedenen Varianten, die den Tools entsprechen, gespeichert werden kann.

Ein Schlüsselbegriff von MLFlow sind die Varianten (im Deutschen wird auch das englische Flavor verwendet), anhand derer Bereitstellungstools die Modelle verstehen können. Damit wird die Arbeit mit allen ML-Bibliotheken ermöglicht, ohne dass Tools in die Bibliothek selbst integriert werden müssen. Aufgrund der Varianten können ...

- ... dasselbe Speicherformat in verschiedenen Systemen verwendet werden;
- ... der Mehraufwand von Kommunikation über Systeme hinweg (Serialisierung und Deserialisierung) vermieden werden;
- ... gemeinsam benutzbare Funktionen verfügbar gemacht werden.

MLFlow hat standardmäßig integrierte Varianten für viele beliebte Machine-Learning-Algorithmen und -Bibliotheken, wie H2O, Keras, MLeap, PyTorch, Scikit-Learn, MLlib, Tensorflow, ONNX (Open Neural Network Exchange), MXNET gluon, XGBoost und LightGBM. Zudem besteht die Möglichkeit, anhand von Python benutzerdefinierte Varianten anzulegen (MLFlow, o. D.).

Modellregistrierung

Durch die Modellregistrierung soll das Problem der Machine-Learning-Modellverwaltung gelöst werden, denn es ermöglicht die Verwaltung des gesamten Lebenszyklus eines ML-Modells mithilfe der folgenden Aspekte:

- **Modell:** Ein Modell geht aus einem Experiment oder einer Ausführung hervor. Experimente und Ausführungen werden anhand von speziellen MLFlow-Protokollierungsmethoden protokolliert. Nach der Protokollierung kann ein Modell in der Modellregistrierung registriert werden.
- **Registriertes Modell:** Dort ist es dann als registriertes Modell mit einem einmaligen Namen, Containerversionen, zugehörigen Übergangsphasen, Modellherkunft und anderen Metadaten registriert.
- **Modellversion:** Registrierte Modelle können eine oder mehrere Versionen haben. Das erste registrierte Modell hat die Version 1. Jedes weitere Modell, das mit diesem Modell registriert wird, bekommt die nächsthöhere Versionsnummer.
- **Modellphase:** Jede einzelne Modellversion kann einer Phase (z. B. Staging, Produktion oder Archiviert) zugeordnet werden. Modelle können von einer Phase in eine andere übertragen werden.
- **Annotationen und Beschreibungen:** Das gesamte Modell, sowie die einzelnen Versionen, können mit einer Beschreibung und anderen relevanten Informationen annotiert werden.

MLFlow kann so, wie folgt, installiert werden:

Code

```
pip install MLFlow
```

Eine einfache Pipeline kann wie folgt aussehen:

Code

```
from mlflow import log_metric, log_param, log_artifact
if __name__ == "__main__":
    # Log a parameter (key-value pair)
    log_param("param1", 5)
    # Log a metric
    log_metric("foo", 1)
    log_metric("foo", 2)
    log_metric("foo", 3)
    # Log an artifact (output file)
```

```
with open("output.txt", "w") as f:  
    f.write("Hello world!")  
log_artifact("output.txt")
```

Dieser Code ...

- ... legt den Wert 5 für den Parameter param1 fest;
- legt die Werte 1, 2, und 3 für den Messwert **foo** fest;
- öffnet die Datei (ein Artefakt) output.txt, um in sie zu schreiben; und
- legt anhand von log_artifact das Artefakt für die Ausführung fest. Das sind die wichtigsten Aspekte, die bei jeder Ausführung des Auftrags als Änderung protokolliert werden müssen.

Diese Daten dienen offensichtlich nur als Beispiel (siehe „Hello, World!“), um zu verdeutlichen, wie die Schritte im Prozess in einem Python-Skript annotiert werden können (GitHub, o. D.-b.).

Pipelinemanagement mit Kubeflow

Kubeflow wurde von Google als Open-Source-Plattform zur Ausführung von TensorFlow geschaffen. Kubeflow ist eine skalierbare und portierbare Machine Learning-Plattform, mit der Bereitstellungen von ML-Workflows auf Kubernetes verwaltet werden können (Kubeflow, o. D.-a). Die Zielgruppe sind Datenwissenschaftler:innen, die Datenpipelines bauen und damit experimentieren. Kubeflow ist auch dafür gedacht, Machine-Learning-Systeme in verschiedenen Umgebungen bereitzustellen, wo Tests, Entwicklung und Produktionsdienste stattfinden. Als Framework für mehrere Cloudanbieter und zur Unterstützung mehrerer Architekturen führt es gesamte ML-Pipelines und die Bereitstellungspipeline aus und bietet spezielle Tools für alle Schritte in der Modellerstellungspipeline, die bereits abgedeckt wurden.

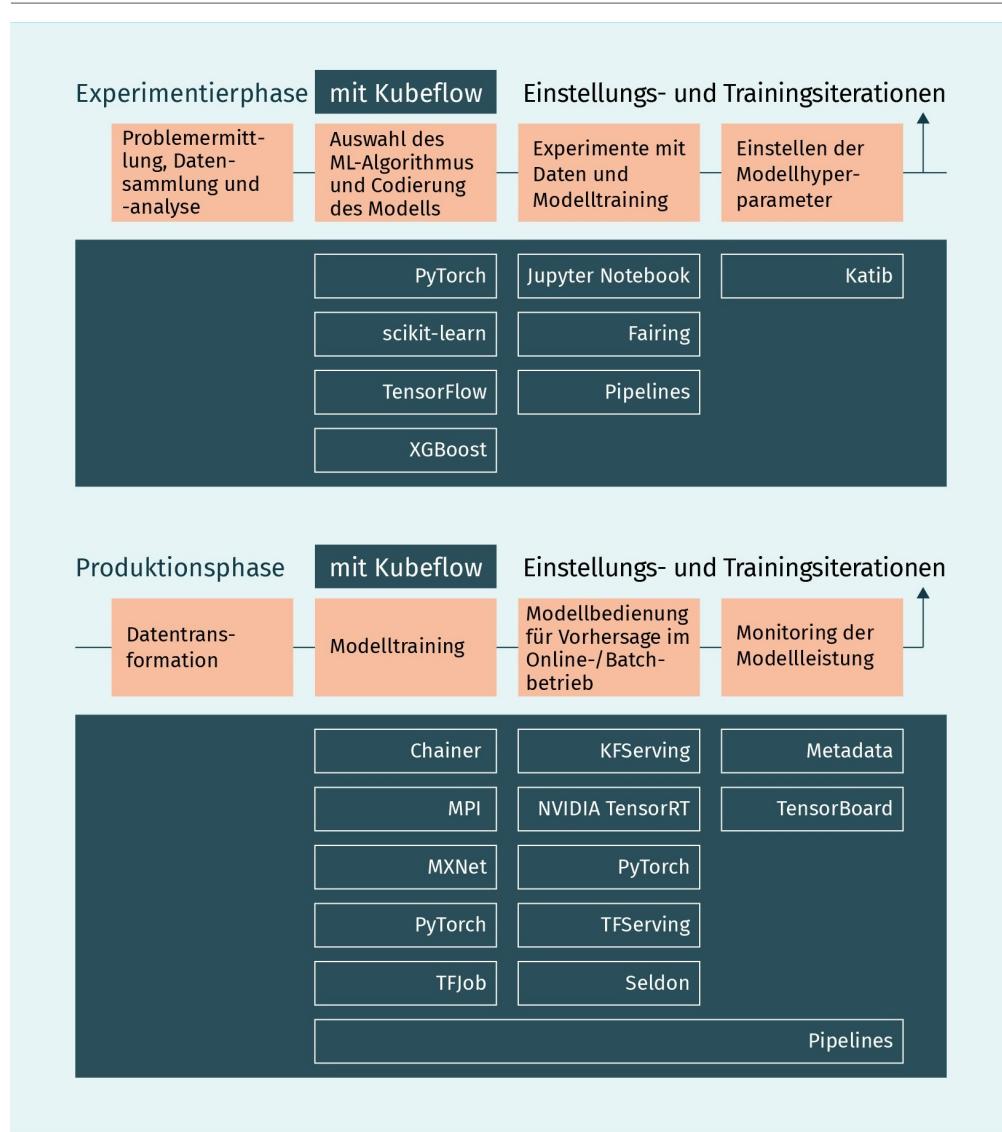
Foo

Foobar, foo, bar und ähnliches werden in der Informatik als Platzhalternamen verwendet.

„Hello, World!“

Ein einfaches Programm, das oft dazu verwendet wird, die einfachste Syntax einer Programmiersprache zu illustrieren.

Abbildung 51: Modellentwicklung und Produktion mit Kubeflow



Quelle: Kubeflow.

Wie im Diagramm ersichtlich, kann Kubeflow mit häufig verwendeten Tools (z. B. PyTorch, scikit-learn, Jupyter Notebooks, TensorBoard usw.) integriert werden. Kubeflow selbst liefert zusätzliche, eigene Komponenten. Eine Komponente ist der Dienst zur Erzeugung und Verwaltung von Jupyter Notebooks, interaktiven Data Sciences und zum Experimentieren mit ML-Workflows. Notebooks können in einer Organisation gemeinsam benutzt werden. Zudem können Notebooks in Pods oder Containern im Cluster statt lokal erstellt werden. Administratorensteuerung ist ebenfalls in Kubeflow integriert, wodurch Standardnotizbuchbilder, die Einrichtung von rollenbasierter Zugriffssteuerung, Geheimnisse und Anmeldeinformationen, um zu regeln, wer auf die Notebooks Zugriff hat, möglich sind. Kubeflow-Pipelines bietet eine Plattform zur Erstellung, Bereitstellung und Verwaltung von mehrstufigen ML-Workflows auf der Basis von Docker-Containern. Eine Pipeline ist in

Kubeflow eine Beschreibung eines ML-Workflows mit allen Komponenten und der Beschreibung ihrer Zusammenarbeit. Sie enthält die Definition von Inputs und Outputs jeder Komponente.

Jede Komponente einer Pipeline ist ein Docker-Image mit eigenständigem Code, der einen Schritt in der Pipeline ausführt. Eine Komponente kann für die Datenvorbereitung, die Datentransformation, das Modelltraining usw. zuständig sein (MLFlow, o. D.).

Die Pipelines sind in Python-Code beschrieben. Es folgt ein Spielzeugmodell einer Pipeline für eine Funktion zur Addition zweier Zahlen. Es sollte beachtet werden, dass der Python SDK für Kubeflow kfp ist. Zunächst wird eine einfache Funktion add definiert. Sie wird einer Komponente hinzugefügt. Dann wird eine Pipeline mit der Bezeichnung calc_pipeline (mit dem Decorator-Element @dsl.pipeline) erstellt. Die Argumente, die an die Pipeline, also an die Funktion, gegeben werden, werden festgelegt, und schließlich wird eine Ausführung für die Pipeline mit folgendem Code erstellt:

Code

```
kfp.Client().create_run_from_pipeline_func().  
import kfp  
import kfp.components as comp  
import kfp.dsl as dsl  
# we define a very simple function, adding 2 numbers def add(a: float,  
b: float) -> float:  
    '''Calculates sum of two arguments''' return a + b  
add_op = comp.func_to_container_op(add)  
  
@dsl.pipeline(  
    name='Calculation pipeline',  
    description='A toy pipeline that performs arithmetic calculations.'  
)  
def calc_pipeline(  
    a='a',  
    b='7',  
):  
    result_task = add_op(a, 4) #Returns a dsl.ContainerOp class instance.  
  
#Specify pipeline argument values  
arguments = { 'a': '7', 'b': '8'}  
#Submit a pipeline run  
kfp.Client().create_run_from_pipeline_func(calc_pipeline, arguments=arguments)
```

Kubeflow hat mehrere Komponenten, die zur Erstellung des ML-Trainings, der Abstimmung der Hyperparameter und zur Bedienung der Workload über mehrere Plattformen verwendet werden können. Hier zeigt sich, dass Kubeflow Lösungen für viele der im vorherigen Abschnitt angesprochenen Themen anbietet.

Machine Learning-Lebenszyklus - von Anfang bis Ende mit Michelangelo

Technische Schulden

Implizierte Kosten zusätzlicher Nacharbeit durch die Wahl einer einfachen und beschränkten Lösung, statt eines nachhaltigeren Ansatzes mit einer längeren Implementierungsphase.

Michelangelo (Hermann & Del Balso, 2017) wurde im Jahr 2015 von Uber entwickelt, um, wie Uber es ausdrückte „versteckte **technische Schulden** in Machine-Learning-Systemen“ anzugehen (Sculley et al., 2015). Michelangelo sollte das Problem der anwendungsspezifischen einmaligen Systeme lösen, das eng mit Machine-Learning-Modellen verbunden ist, einem Ansatz, der, wie bereits besprochen, nicht mitwachsen kann.

Michelangelo basiert auf transaktionalen und protokollierten Daten und unterstützt Offlinevorhersagen (im Batchbetrieb) und Onlinevorhersagen (gestreamte Daten). Offlinevorhersagen werden zur Ausführung des Modells mit dem gesamten Datensatz verwendet, die Ergebnisse werden nach der Ausführung im Batch ausgegeben. Onlinevorhersagen hingegen werden für laufende Abfragen verwendet. Offlinevorhersagen gründen auf Spark-Aufträgen in Containern für Batchvorhersagen, während Onlinevorhersagen in einem Vorhersage-Cluster bereitgestellt werden, sodass ein Load Balancer die Last auf verschiedene Maschinen verteilen kann. Unter Lastenausgleich wird der Vorgang verstanden, bei dem eine Reihe von Aufgaben auf eine Reihe von Ressourcen verteilt werden. Das Ziel dabei ist, die Gesamtarbeit effizienter zu gestalten. Bei jedem Experiment werden für die Modellverwaltung relevante Daten gespeichert (z. B. Runtimewerte des Trainers, Modellkonfiguration, Herkunft, Verteilung und relative Bedeutung von Funktionen, Messwerte zur Modellbeurteilung, Standardbeurteilungsdiagramme, erlernte Parameterwerte und zusammenfassende Statistiken). Mit Michelangelo können mehrere Modelle im selben bedienenden Container bereitgestellt werden, was den Übergang von alten zu neuen Modellversionen und gleichzeitiges A/B-Testen der Modelle ermöglicht. Michelangelo unterstützt sowohl Online- als auch Offlinemodele. Die Daten werden von der Datenaufbereitungspipeline mithilfe von Push in die Funktionsspeichertabellen und Trainingsdaten-Retrositories übertragen. Die Pipelines verwenden Apache Kafka, eine quelloffene Streamverarbeitungssoftware, und stellen eine Verbindung zum Data Lake des **Hadoop**-Dateisystems her. Zur Datenaufbereitung wird entweder **Apache Spark** oder SQL verwendet. Modelltraining wird im Batchbetrieb anhand eines der verschiedenen unterstützten Modelle durchgeführt: Entscheidungsbaum, lineare und logistische Modelle, unüberwachte Modelle (K-Means), Zeitreihenmodelle und Deep Neural Networks. Die Modellkonfiguration legt Modelltyp, Hyperparameter und Referenzen von Datenquellen fest und berechnet Ressourcenanforderungen. Dann wird es auf einem Cluster ausgeführt. Nachdem das Modell trainiert ist, werden Leistungswerte im Modellbewertungsbericht gespeichert. Die ursprüngliche Konfiguration, die erlernten Parameter und der Beurteilungsbericht werden zur Analyse und Bereitstellung im Modellrepository gespeichert. In allen Modelltypen wird die Hyperparametersuche unterstützt. Im NoSQL-Datenbankverwaltungssystem mit verteiltem, breitem Spaltenspeicher, Cassandra, wird für jedes trainierte Modell ein versioniertes Objekt im Modellrepository zur Beurteilung gespeichert. Berichte können im Dashboard visuell mit Standardmesswerten für Genauigkeit und Funktionsberichte dargestellt werden. Modelle können dann auf verschiedene Arten bereitgestellt werden: offline; in einem Container mit Ausführung in einem Auftrag in Spark zur Generierung von Batchvorhersagen; als Dienstcluster für Vorhersage im online Modus; oder als eingebettete Bibliothek, die selbst in einem Dienst eingebettet ist. Sind die Modelle bereitgestellt und vom bedienenden Container geladen, werden sie zur Generierung von Vorhersagen aufgrund von Funktionsdaten aus der Datenpipeline oder direkt aus dem Client-Dienst eingesetzt. Mehrere Modelle können in einem Container bereitgestellt werden.

Messwerte zur Überwachung und zur Leistung werden protokolliert. Die ML-Pipelineserialisierung von Spark wird von der Plattform mit einer zusätzlichen Schnittstelle für die online Bedienung verwendet. Dadurch wird eine online Bewertungsmethode mit einem Beispiel hinzugefügt, die leichtgewichtig ist und **Service-Level-Vereinbarungen (SLAs)** handhaben kann, die in kritischen Fällen, wie Betrugserkennung und -prävention, wichtig sind.

Service-Level-Vereinbarungen (SLA)
Vereinbarung zwischen einem Serviceanbieter und einem Kund:in.

Schlussbemerkung

In diesem Abschnitt wurden die Schwierigkeiten, die aus der Sicht von MLOps auftreten, wenn Machine Learning in Betrieb genommen wird, abgedeckt. Die behandelten Tools weisen unterschiedliche Stärken und verschiedene Ansätze auf. MLFlow ist beispielsweise eng mit Spark integriert (und wurde von derselben Firma entwickelt). MLFlow wird hauptsächlich als Modellverwaltungsbibliothek verwendet. Es hat ein paar Optionen für die Bereitstellung, die über Airflow ablaufen muss. Kubeflow ist eine integriertere Lösung, die auf Kubernetes aufbaut und mehrere quelloffene Bausteine integriert.

Dazu ist die separate Konfiguration der benötigten Elemente notwendig. Zuletzt wurde Michelangelo als Beispiel einer End-to-End-Lösung behandelt. Es integriert die meisten Komponenten mit strukturiertem Zugriff auf die, die nicht integriert sind. Dadurch müssen einige Entscheidungen getroffen werden, wodurch Michelangelo weniger Flexibilität bietet und dafür komplexer wird. Jedes dieser Systeme muss in einer Organisation mit den notwendigen Hardware- und Softwareressourcen eingerichtet und von einem erfahrenen Supportteam unterstützt werden.

5.4 Cloudbasierte Lösungen

Es wurden bereits einige der Schwierigkeiten erörtert, die durch die Inbetriebnahme von Machine Learning entstehen, sowie die Entwicklung von MLOps und die Tools, die einige, viele oder gar alle dieser Schwierigkeiten beheben. Dabei wurde davon ausgegangen, dass diese Tools in die bestehende Infrastruktur einer Organisation integriert werden und in einigen Fällen Hybridlösungen ermöglichen, bei denen für manche Aufgaben auf **Infrastruktur in der Cloud** zugegriffen wird. Der Aufbau und die Instandhaltung einer firmeneigenen Infrastruktur ist weder einfach noch kostenlos. Es müssen das entsprechende „Knowhow“ und interne Ressourcen vorhanden sein. Zudem muss für die Wartung der Hardware- und Softwareinfrastruktur und deren Skalierbarkeit gesorgt werden. Manche Organisationen wollen diese Kosten vermeiden, und das wachsende Angebot an ML-Cloud Services macht dies möglich. Mit ML-Cloud Services können Arbeitsmodelle mit einem relativ kleinen Team gebaut, überwacht, bereitgestellt und für Vorhersagen und Analysen genutzt werden.

Cloudinfrastruktur
Cloud Computing ist die bedarfsabhängige Verfügbarkeit von Computersystemressourcen, vor allem Datenspeicher und Rechenleistung, ohne die aktive Verwaltung durch Benutzer:innen.

Machine Learning as a Service (MLaaS)

Unter dem Begriff „Machine Learning as a Service“ werden verschiedene cloudbasierte Plattformen zusammengefasst. Sie decken die meisten Infrastrukturaspekte, die im Rahmen von MLOps benötigt werden, ab: Datenvorbereitung, Modelltraining, Modellbewer-

Remoteprozeduraaufruf (RPC)

Aufruf eines Computerprogramms, das zur Ausführung einer Prozedur auf einem anderen Computer im Netz führt.

tung, Bereitstellung und Monitoring. Das Output kann über REST-APIs und andere Formen der internen Kommunikation, wie **RPC** und andere Protokolle an die interne IT-Infrastruktur weitergegeben werden. Im Jahr 2020 waren die führenden Anbieter von MLaaS in der Cloud Amazon Machine Learning Services, Azure Machine Learning, Google Cloud AI und IBM Watson (IBM, o. D.-c). Ihr Ziel bestand darin, Modelltraining und -bereitstellung zu beschleunigen (AltexSoft, 2019).

Die Amazon Machine Learning Services sind auf zwei Ebenen erhältlich: Predictive Analytics mit Amazon ML und SageMaker, das hauptsächlich für Datenwissenschaftler:innen konzipiert wurde (Amazon, o. D.-b). Amazon Machine Learning für predictive Analytics ist eine weitestgehend automatisierte Lösung für Arbeitsgänge nach Terminplan. Es kann Daten aus mehreren Quellen, z. B. Amazon RDS, Amazon Redshift und .csv Dateien, laden. Die Datenvorbereitung läuft automatisch ab. Amazon ML versucht zu erkennen, welche Felder kategorial und welche numerisch sind. Benutzer:innen müssen dabei keine Entscheidung zu Methoden zur weiteren Datenvorbereitung, wie Dimensionalitätsreduktion, treffen. Die Vorhersagemöglichkeiten von Amazon ML sind jedoch auf binäre oder mehrklassige Klassifikationen und Regression begrenzt. Unüberwachte Lernmethoden werden nicht unterstützt, und Benutzende müssen eine Zielvariable auswählen, um sie im Trainingssatz zu kennzeichnen. Da es sich an Nicht-Datenwissenschaftler:innen richtet, müssen Benutzende keine Kenntnisse über ML-Methoden haben. Diese werden aus den gelieferten Daten automatisch ausgewählt. Es stellt eine automatisierte, wenn auch begrenzte, Lösung zur Verfügung.

Für fortgeschrittene Anwender:innen, die mehr Flexibilität und Komplexität benötigen, gibt es SageMaker. Amazon SageMaker ist eine Machine-Learning-Umgebung mit Tools zur schnellen Modellerstellung und -bereitstellung. Jupyter ist in SageMaker zur Datenerkundung und -analyse integriert, ohne die Notwendigkeit der Serververwaltung (Amazon, o. D.-b).

Machine Learning mit Amazon SageMaker

Um zu zeigen, wie Cloud-Lösungen alle Tools zur Erstellung einer End-to-End-Pipeline für Machine Learning liefern können, konzentrieren wir uns auf den Amazon SageMaker. Die Produkte anderer Anbieter verfügen über ähnliche Möglichkeiten, und unterscheiden sich in den Details. Der Amazon SageMaker ist ein vollständig verwalteter Dienst, der alle Tools für die Erstellung, das Training und die Bereitstellung von ML-Modellen liefert. Alle Schritte einer Machine-Learning-Pipeline können mit SageMaker ausgeführt werden (Amazon, o. D.-c). Hier werden sie noch einmal aufgeführt: das Problem verstehen, Daten sammeln und vorbereiten, die explorative Datenanalyse, mit Funktionen und Modellen experimentieren, das Modell trainieren und auswählen, die Versionen verwalten, das Modell bereitstellen, die Leistung überwachen und mit anderen Systemen integrieren.

Voraussetzungen

Da SageMaker auf den Amazon Web Services basiert, muss AWS zur Verwendung vorhanden sein oder zu diesem Zweck eingerichtet werden. Anschließend müssen ein IAM-Administrator und eine IAM-Administratorengruppe erstellt werden. AWS ist ein leistungsstarkes, jedoch nicht immer intuitives System und die Einarbeitungszeit kann dauern. Größere

Organisationen verfügen daher über dedizierte Prozesse und Teams. Wenn AWS in einem Unternehmen eingerichtet wird, muss eine IAM-Benutzerrolle mit Administratorrechten angelegt werden. Die Arbeit mit AWS-Diensten ist ein Thema, das ganze Bücher füllen könnte, daher sollten sich Interessierte ein wenig damit vertraut machen (Amazon, o. D.-e).

Problemverständnis

Zurzeit gibt es kein automatisiertes System, das uns diese Phase erleichtern kann. Dazu wäre ein weit entwickeltes KI-System notwendig. SageMaker verfügt über einige automatische Tools, wie den SageMaker Autopilot, die jedoch auf wenige Domänen begrenzt sind. Dazu gehören: Autopilot, Regression, binäre und mehrklassige Klassifikation.

Datensammlung, -bereinigung und -aufbereitung

Einen Schritt, den wir bisher noch nicht berücksichtigt haben, der jedoch bei der Datenverwaltung in einer Organisation wichtig ist, ist die Datenquelle und der Datenspeicher. Als vollständig verwaltete Lösung, die auf den vielen Diensten von AWS basiert, integriert der SageMaker eigene Funktionen zur Datenspeicherung. Dazu ist ein „**Data Lake**“ nötig. In der Regel kann auf Daten aus verschiedenen Quellen zugegriffen werden. In einer AWS-Umgebung werden die Rohdaten zunächst in einem S3-Bucket gespeichert. Amazon Simple Storage Service (S3) hat Objektspeicher über eine Webdienstschnittstelle. Auch Vorbereitungscode kann in S3 gespeichert werden. SageMaker greift dann dort darauf zu. In SageMaker findet die Datenvorbereitung in einem **Jupyter** Notebook, d. h. in einer Notebooksinstanz statt. Notebooks werden zum Abrufen, Erkunden und Vorbereiten von Datensätzen für das Modelltraining abgerufen. Zunächst muss eine Notebooksinstanz in SageMaker gestartet werden.

Abbildung 52: SageMaker Konsole

Name	Instance	Creation time	Status	Actions
p3-2xl-oregon	ml.p3.2xlarge	Jan 15, 2019 19:21 UTC	InService	Open Jupyter Open JupyterLab
m5xl1st-oregon	ml.m5.xlarge	Nov 19, 2018 00:16 UTC	InService	Open Jupyter Open JupyterLab
p2xl-oregon	ml.p2.xlarge	Mar 15, 2018 23:37 UTC	Stopped	Start

Quelle: Pedori (2020).

Dann können ein Name, eine Instanzart und andere Einstellungen ausgewählt werden. In AWS muss immer eine IAM-Rolle erstellt werden. In unserem Fall muss der Zugriff auf das Notebook eingerichtet werden.

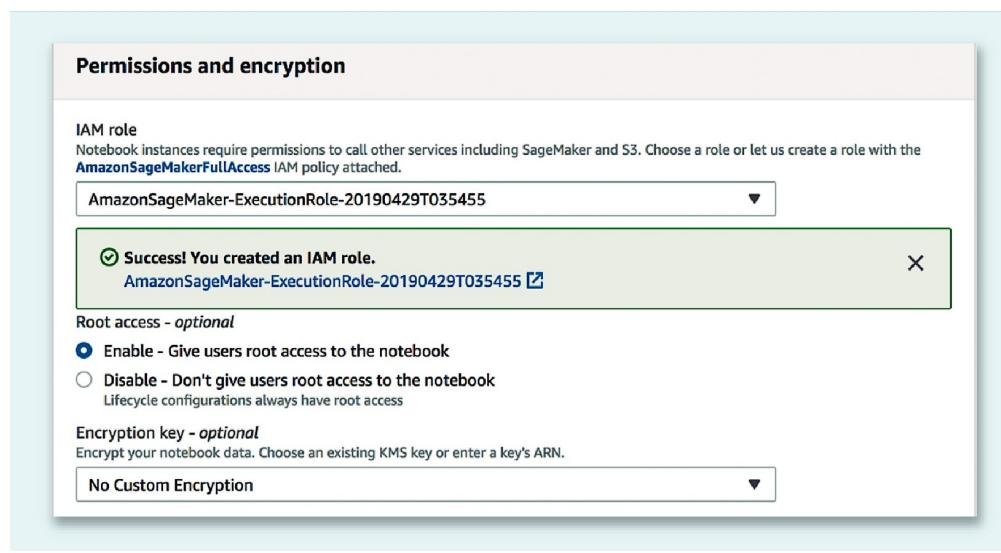
Data Lake

Ein System oder Datenrepository, in dem Daten im natürlichen, unveränderten Format, meistens als Blobs oder Dateien, gespeichert sind.

Jupyter

Eine Open-Source-Webanwendung, mit der Dokumente erstellt und geteilt werden können, die Betriebscode, Gleichungen, Visualisierungen und Beschreibungen enthalten.

Abbildung 53: Zugriff auf IAM-Rollen



Quelle: Pedori (2020).

Anschließend kann auf die Notebookinstanz zugegriffen und Jupyter geöffnet werden. Aus dem Notebook können alle Schritte zur Datenvorbereitung ausgeführt werden. Dazu wird ein AWS Glue verwendet und eine SparkML-Funktion zur Vor- und Nachbereitung ausgeführt. AWS Glue ist ein serverloser Datenaufbereitungsdienst, mit dem Daten extrahiert, bereinigt, aufgebessert, normalisiert und geladen werden können. Das heißt, dass Extraktion, Transformation und Laden in einer grafischen Benutzeroberfläche stattfinden können. SparkML ist eine hochwertige API, die die Erstellung und Einstellung von praktischen ML-Pipelines aus dem Apache-Spark-Projekt unterstützt. Es können mehrere Datenvorbereitungsschritte programmatisch oder manuell eingerichtet werden. Der standardmäßige Python-Client für AWS-Dienste heißt boto (boto, 2020). Im folgenden Code wird gezeigt, wie ein Auftrag in AWS aus einem Notebook ausgeführt wird. Die Auftragserstellung wird dabei nicht gezeigt.

Code

```
import time
import boto3

glue = boto3.client('glue')

# Run the job in AWS Glue
job_name='preprocessing-cars'
response = glue.start_job_run(JobName=job_name)
job_run_id = response['JobRunId']
print('{}\n'.format(response))

# Check on the job status every 30 seconds
job_run_status = glue.get_job_run(JobName=job_name, RunId=job_run_id)
['JobRun']['JobRunState']
```

```

while job_run_status not in ('FAILED', 'SUCCEEDED', 'STOPPED'):
    job_run_status = glue.get_job_run(JobName=job_name, RunId=job_run_id)
    ['JobRun'] ['JobRunState']
    print (job_run_status)
    time.sleep(30)

```

Explorative Datenanalyse (EDA)

In einem Jupyter Notebook sind die EDA-Schritte dieselben wie auf einer lokalen Maschine oder in On-Premises-Umgebungen. Auch derselbe Python-Code wird dabei verwendet. Dieser Schritt muss nicht angepasst werden. Der einzige Unterschied ist, dass das Notebook, auf dem gearbeitet wird, auf AWS mit SageMaker gehostet wird.

Experimentieren, Training und Versionsverwaltung

Die Tools von SageMaker kommen erst richtig zum Einsatz, wenn das Experimentieren mit Modellen, Funktionen und Hyperparametern beginnt. Alle Vorgänge werden als Trainingsauftrag entweder von der Benutzeroberfläche oder aus dem Python SDK ausgeführt. SageMaker hat einige integrierte Modelle, die auf ein Problem angewendet werden können (Amazon, o. D.-f.). Überwachtes und unüberwachtes Lernen, Textanalyse und Bildverarbeitung sind dadurch abgedeckt. Im Jahr 2020 waren folgende Algorithmen verfügbar: BlazingText Algorithmus, DeepAR-Prognosen, Faktorisierungsmaschinen, Bildklassifizierungsalgorithmus, Algorithmus zur Analyse von IP Nutzungsmuster, k-Means-Algorithmus, kNN-Algorithmus, lineare Diskriminanzanalyse (LDA), Algorithmus für lineares Lernen, Algorithmus für neuronale Themenmodellierung (NTM), Object2Vec, Objekterkennungsalgorithmus, Algorithmus für die Hauptkomponentenanalyse, RCF-Algorithmus, semantische Segmentierung, SageMaker Sequence-to-Sequence (seq2seq) und XGBoost-Algorithmus (eXtreme gradient boosting). Ein Algorithmus kann auch von Benutzer:innen erstellt und in Docker-Container verpackt werden, indem entweder ein vorgefertigtes Docker-Image erweitert wird oder ein eigenes angepasst bzw. erstellt wird. SageMaker ist sehr gut mit SparkML zur Funktionsauswahl und Hyperparameteroptimierung integriert. SparkML wird dabei für die meisten Aufgaben eingesetzt. Eine der wichtigsten Funktion von SageMaker ist SageMaker Experiments, mit dessen Hilfe ML-Experimente organisiert, verfolgt, verglichen und beurteilt werden können. SageMaker Experiments führt die Begriffe „Versuch“ und „Experiment“ ein. Ein Versuch ist die Sammlung von Trainingsschritten in einem einzelnen Trainingsauftrag, der aus Vorbereitung, Training und Bewertung besteht und mit Metadaten angereichert wird. Ein Experiment ist eine Sammlung von Versuchen, d. h. eine Gruppe von verwandten Trainingsaufträgen. In SageMaker Experiments können Experimente erstellt und ihnen können Versuche hinzugefügt werden sowie Analytics an Versuchen und Experimenten ausgeführt werden.

Zugriff auf Experiments kann von der Benutzeroberfläche und über den Python SDK, der Protokollierungs- und Analyse-APIs enthält, erfolgen. SageMaker Experiments ist im SageMaker Autopilot integriert, sodass bei der Ausführung eines Auftrags in Autopilot automatisch ein Experiment mit Versuchen für die verschiedenen Kombinationen von Komponenten, Hyperparametern und Artefakten erstellt wird. Zu Trainingsaufträgen und zu Aufträgen in Autopilot muss jeweils nur ein Parameter in den Python-Code im SDK eingefügt werden, durch den das Experiment definiert wird. Das Ergebnis ist eine Sammlung

von in Echtzeit aktualisierten Experimenten, die visuell und programmatisch verglichen und beurteilt werden können, um das bis zu dem Zeitpunkt beste Modell auszuwählen. Es gilt zu bemerken, dass in Experiments zwar Versionsverwaltung von Modellen, nicht aber von Daten und Code möglich ist. Das muss separat geschehen. Notebooks in SageMaker unterstützen Git, d. h. Datenversionsverwaltungssysteme können im Workflow erstellt oder in den Workflow integriert werden. Der gleiche Ablauf kann fürs letzte Training verwendet werden. Da das Python-SDK verfügbar ist, kann dies programmatisch festgelegt und automatisch ausgeführt werden. SageMaker unterstützt zusätzliche Modellvalidierung mit offline Daten oder Betriebsdaten zum A/B-Testen.

Bereitstellung

Wenn das gewählte Modell trainiert ist, kann es auf zwei Arten im SageMaker bereitgestellt werden. Es können entweder persistente Endpunkte eingerichtet werden, um jeweils eine Vorhersage zu erhalten, wozu SageMaker Hosting Services verwendet wird. Oder es wird SageMaker Batch Transform verwendet, um Vorhersagen für den gesamten Datensatz abzufragen. Für SageMaker Hosting Services muss ein HTTPS-Endpunkt erstellt und SageMaker zugänglich gemacht werden. Damit kann er für Vorhersagen verwendet werden. Diese Endpunkte, die auf einzelnen AWS-Konten angelegt sind, sind nicht öffentlich. Wenn nötig, kann öffentlicher Zugriff über AWS Lambda und den Amazon API-Gateway ermöglicht werden (Amazon, o. D.-g.).

Monitoring

Die Vorteile der AWS-Umgebung kommen bei der Bereitstellung, während der Produktion und durch den Monitoring-Support zum Tragen. SageMaker hat ein Modellüberwachungstool, mit dem die Qualität der Modelle in Echtzeit überwacht werden kann. Der Amazon SageMaker Model Monitor ermöglicht es, ein automatisches Warnsystem einzurichten, das Warnungen ausgibt, wenn Abweichungen in der Modellqualität, wie Datendrift und andere Unregelmäßigkeiten, auftreten. Amazon CloudWatch Logs sammelt Protokollierungsdateien und sendet Benachrichtigungen aus, wenn die Modellqualität bestimmte, festgelegte Schwellwerte erreicht hat. Über AWS CloudTrail werden die Protokollierungsdateien in einem Amazon S3-Bucket gespeichert (Amazon, o. D.-h.). Modellabweichungen und andere potenzielle Probleme können durch proaktive Maßnahmen früh erkannt werden. So könnte der Modell Drift entgegen gewirkt werden. Mit der Funktionalität von SageMaker können mehrere Modelle oder Modellversionen nach demselben Endpunkt mit Produktionsabweichungen getestet werden. Für diese Tests können bestimmte Datenverkehrsverteilungen festgelegt und bestimmte Varianten aufgerufen werden. Auch A/B-Tests sind möglich. Anschließend kann die Modellperformance beurteilt und der Verkehr erhöht werden, um zum besten Modell zu gelangen. Wenn nötig, können Modelle mit denselben Funktionen wie beim ersten Training und der Auswahl neu trainiert werden.

Vorteile und Nachteile von ML-Cloud-Lösungen

Wie am Beispiel von Amazon SageMaker gezeigt, kann mit Machine Learning as a Service von Cloudanbietern die komplette Modellentwicklungs- und -produktionspipeline abgewickelt werden. Sie bieten entweder On-Premises-Umgebungen oder Unterstützung mit

den verschiedensten Lösungen an. Ihr Ziel sind skalierbare Produktionsumgebungen. Daher liefern sie fertige Funktionalität, z. B. für Monitoring und Skalierbarkeit. Dadurch kann ein Team klein und mit wenigen Ressourcen anfangen und bei Bedarf wachsen.

Aber es gibt auch Nachteile: Obwohl die meisten Cloudanbieter ihre Lösungen auf Open-Source-Software anbieten (Jupyter, SparkML, Python, PyTorch, Docker, Kubernetes, Git usw.), wird diese oft angepasst oder sehr eng mit den proprietären Lösungen verbunden. Das stellt an sich kein Problem dar. Die Portierung wird jedoch beeinträchtigt. Sollte beispielsweise die Infrastruktur von SageMaker gelöst werden, müssten die möglicherweise verwendeten, integrierten Algorithmen zum Modelltraining neu implementiert werden oder neuer Code geschrieben werden. Der Einsatz von Cloudanbietern führt zu zusätzlichen Kosten, genau wie die Verwaltung der Infrastruktur. Eine SageMaker EC2-Instanz auf AWS kostet beispielsweise 40 % mehr als eine Barebone-AWS EC2-Instanz. Dies ist kein schlechter Kompromiss für kleine Organisationen, die weder über IT-Support noch über die notwendigen Ressourcen verfügen. Wenn Letzteres allerdings vorhanden ist, wären die Kosten ein unnötiger Mehraufwand. Und schließlich könnten Lösungen für eine bestimmte Architektur entwickelt werden, bei der Annahmen und Tools an den Anbieter angepasst werden. Dies bietet sich an, wenn nicht genügend Erfahrung vorhanden ist, denn die Architekturen der Cloudanbieter beruhen auf den Best Practices. Der Nachteil davon ist, dass das System weniger flexibel ist. Durch diese Aspekte kommt es immer mehr zum Anbieter-Lock-In. Die ML-Lösung hängt stark vom Anbieter ab und Änderungen können sehr kostspielig werden, und die Kosten steigen mit zunehmender Komplexität des Machine-Learning-Systems. Im Allgemeinen kommt eine wachsende Organisation irgendwann an den Punkt, an dem eine eigene Infrastruktur sinnvoll ist. Zu frühe Eigenständigkeit ist unnötig teuer. Wenn sich ein Unternehmen zu spät vom Anbieter löst, können ebenfalls Kosten anfallen, aber auch Risiken auftreten. Die Vorteile von Cloud-Lösungen für Machine Learning lassen sich wie folgt zusammenfassen:

- skalierbare Produktionsumgebungen,
- integriertes Monitoring,
- Mitlieferung von Best Practices sowie
- Lösung mehrerer Probleme auf einmal.

Und die zusammengefassten Nachteile von ML-Cloud-Lösungen:

- **Anbieter-Lock-In** und eingeschränkte Portierung,
- höhere Kosten sowie
- proprietäre Lösungen.



ZUSAMMENFASSUNG

Diese Lektion führte in den Entwicklungsprozess von ML-Modellen ein, indem explorative, iterative und nichtdeterministische Eigenschaften behandelt wurden. Zudem wurden Schwierigkeiten in Bezug auf Datenqualität und -erkundung, Entwicklung, Tests und die Bereitstellung von Machine-Learning-Modellen eruiert. Anschließend wurde dargelegt, wie

Anbieter-Lock-In

Die Abhängigkeit eines Unternehmens von einem Anbieter für Produkte und Dienstleistungen, weil beim Wechsel auf einen anderen Anbieter zu große Wechselkosten entstehen würden.

anspruchsvoll es sein kann, ein Machine-Learning-System in Betrieb zu nehmen und zu halten, wobei auch auf Überschneidungen und Abweichungen zur Entwicklung herkömmlicher Software eingegangen wurde.

Darüber hinaus wurden Unterschiede zwischen der Versionsverwaltung und dem Testen von ML-Systemen herausgearbeitet. Schließlich wurden mögliche Lösungen für potenzielle Probleme dargelegt.

ANHANG

LITERATURVERZEICHNIS

- AltexSoft. (2019, 27. September). *Comparing machine learning as a service: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson* [Machine Learning as a Service im Vergleich: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson]. <https://www.altexsoft.com/blog/data-science/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/>
- Amazon. (o. D.-a). *What is DevOps? [Was ist DevOps?]*. Amazon Web Services. <https://aws.amazon.com/devops/what-is-devops/>
- Amazon. (o. D.-b). *Amazon SageMaker [Amazon SageMaker]*. Amazon Web Services. <https://aws.amazon.com/sagemaker/>
- Amazon. (o. D.-c). *Machine learning on AWS [Machine Learning auf AWS]*. Amazon Web Services. <https://aws.amazon.com/machine-learning/>
- Amazon. (o. D.-d). *Set up Amazon SageMaker[Amazon SageMaker einrichten]*. Amazon Web Services. <https://docs.aws.amazon.com/sagemaker/latest/dg/gs-set-up.html>
- Amazon. (o. D.-e). *Hands-on tutorials[Praktische Anleitungen]*. Amazon Web Services. <https://aws.amazon.com/getting-started/hands-on/>
- Amazon. (o. D.-f). *Use Amazon SageMaker built-in algorithms[Die integrierten Algorithmen von Amazon SageMaker im Gebrauch]*. Amazon Web Services. <https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html>
- Amazon. (o. D.-g). *Deploy a model in Amazon SageMaker[Bereitstellen von Modellen in Amazon SageMaker]*. Amazon Web Services. <https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-deployment.html>
- Amazon. (o. D.-h). *Monitor Amazon SageMaker[Monitoring in Amazon SageMaker]*. Amazon Web Services. <https://docs.aws.amazon.com/sagemaker/latest/dg/monitoring-overview.html>
- Anji, K. (2020, 22. September). *12 factor app principles and cloud-native microservices[Die Prinzipien der Twelve-Factor-App und der cloudnativen Microservices]*. DZone. <https://dzone.com/articles/12-factor-app-principles-and-cloud-native-microser>
- AOE. (o. D.). *Agile methods & processes in companies[Agile Methoden und Prozesse im Unternehmen]*. <https://www.aoe.com/en/agile.html>
- Apache Maven Project. (o. D.). *Welcome to Apache Maven [Willkommen bei Apache Maven]*. Apache. <https://maven.apache.org/>

- Apache Mesos. (o. D.). *What is Mesos? A distributed systems kernel [Was ist Mesos? Ein Kernel für verteilte Systeme.]*. <http://mesos.apache.org/>
- API Evangelist. (2012). *The secret to Amazon's success internal APIs [Das Erfolgsgeheimnis der internen APIs von Amazon]*. <https://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/>
- Arora, S. (2020, 1. Juli). *Ansible vs. Puppet: The key differences to know [Ansible im Vergleich mit Puppet: Die wichtigsten Unterschiede]*. Simplilearn Solutions. <https://www.simplilearn.com/ansible-vs-puppet-the-key-differences-to-know-article>
- Atlassian Bitbucket. (o. D.). *Git checkout [Git Checkout]*. Atlassian. <https://www.atlassian.com/git/tutorials/using-branches/git-checkout>
- AVI Networks. (o. D.). Single point of failure definition [Definition von Single Point of Failure]. <https://avinetworks.com/glossary/single-point-of-failure/>
- Bach, J. (2003). *Exploratory testing explained [So geht exploratives Testen]*. Satisfice. <http://satisfice.us/articles/et-article.pdf>
- Ballard, G., & Howell, G. (2003). Lean project management [Lean Projektmanagement]. *Building Research & Information*, 31(2), 119–133.
- Baucherel, K. (2019). Scrum in rugby [Scrum beim Rugby]. Pixabay. <https://pixabay.com/photos/rugby-scrum-heineken-cup-saracens-4498375/>
- Basu, A. (2015). *Software quality assurance, testing and metrics* [Qualitätssicherung, Testen und Messungen bei der Softwareerstellung]. PHI Learning.
- Beck, K. (1999). Embracing change with extreme programming [Extreme Programming und der Umgang mit Änderungen]. *Computer*, 32(10), 70–77.
- Beck, K. (2014). *Test-driven development: By example* [Testgetriebene Entwicklung an Beispielen erklärt]. Addison-Wesley.
- Bentley, C. (2010). *PRINCE2: A practical handbook* (3. Aufl.) [PRINCE2: Ein praktisches Handbuch]. Routledge.
- Berg, J., Graham, M., & Whitney, K. (1981). *Database architectures, a feasibility workshop report* [Datenbankarchitekturen: Bericht aus einem Machbarkeitworkshop]. Handelsministerium der Vereinigten Staaten. <https://babel.hathitrust.org/cgi/pt?id=mdp.39015077587742&view=1up&seq=53>
- Berners-Lee, T. (1996). *Hypertext transfer protocol – HTTP/1.0* [Hypertext Transfer-Protokoll - HTTP/1.0]. <https://tools.ietf.org/html/rfc1945>
- Bitkeeper. (o. D.). *Bitkeeper Homepage*. <https://www.bitkeeper.org>

- Bourne, S. R. (1978). Unix time-sharing system: The Unix shell [Das Unix Time-Sharing-System: Die Unix-Shell]. *The Bell System Technical Journal*, 57(6), 1971–1990.
- boto. (2020). *boto. A Python interface to Amazon Web Services*[Eine Python-Schnittstelle auf Amazon Web Services]. Cloudhackers. <https://boto.cloudhackers.com/en/latest/>
- Boyd, M. (2014, 21. Februar). *Private, partner or public: Which API strategy is best for business?* [API-Strategieen fürs Unternehmen: Welche ist am besten?]. Programmableweb. <https://www.programmableweb.com/news/private-partner-or-public-which-api-strategy-best-business/2014/02/21>
- Breck, E., Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2019). Data validation for machine learning [Datenvielfältigung für Machine Learning]. *Proceedings of the 2nd SysML conference*. MLSys.
- Bright, P. (2016, 18. August). *PowerShell is Microsoft's latest open source release, coming to Linux, OS X* [PowerShell auf Linus und OS X - der neueste Open-Source-Beitrag von Microsoft]. Ars Technica. <https://arstechnica.com/information-technology/2016/08/power-shell-is-microsofts-latest-open-source-release-coming-to-linux-os-x/>
- Brikman, Y. (2016, 26. September). *Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation* [Warum Terraform benutzen und nicht Chef, Puppet, Ansible, SaltStack oder CloudFormation]. Gruntwork. <https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language models are few-shot learners* [Few-Shot-Lernen: Sprachmodelle lernen schnell]. arXiv. <https://arxiv.org/abs/2005.14165v4>
- Brownlee, J. (2018, 16. August). *Improving how we work through physical Kanban boards.* [Besseres Arbeit mit Kanban-Boards] Kalibrr Design. <https://medium.com/kalibrr-design/improving-how-we-work-through-physical-kanban-boards-74779327bcec>
- Brownlee, J. (2019, 12. August). *A gentle introduction to concept drift in machine learning* [Concept Drift beim Machine Learning: Eine einfache Einführung]. Machine Learning Mastery. <https://machinelearningmastery.com/gentle-introduction-concept-drift-machine-learning/>
- Buchanan, C. (2019, 12. Juni). *The ideal DevOps team structure*[Die ideale Teamstruktur in der DevOps-Kultur]. GitLab. <https://about.gitlab.com/blog/2019/06/12/devops-team-structure/>
- Carilli, J. F. (2013). *Transitioning to Agile: Ten success strategies* [Zehn Strategien zum Erfolg beim Umstieg auf Agile]. Vortrag. Project Management Institute Global Congress.

Chappell, D. (2008). *What is application lifecycle management? [Was ist Application Lifecycle Management?]*. Wayback Machine. <https://web.archive.org/web/20141207012857/http://www.microsoft.com/global/applicationplatform/en/us/RenderingAssets/White%20papers/What%20is%20Application%20Lifecycle%20Management.pdf>

Chef. (o. D.). *Chef Homepage*. <https://www.chef.io/>

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too [Kontinuerliche Lieferung: Riesenvorteile, aber halt auch Nachteile]. *IEEE Software*, 32(2), 50–54.

Chollet, F. (2017). *Deep learning with Python [Deep Learning mit Python]*. Manning.

CircleCI. (o. D.). *CircleCI Homepage*. <https://circleci.com>

Clarke, S. (2004, 1. May). *Measuring API usability [API-Usability messen]*. Dr. Dobb's. <https://www.drdobbs.com/windows/measuring-api-usability/184405654>

Cockburn, A., & Williams, L. (2001). *The costs and benefits of pair programming: Extreme programming examined* [Die Vor- und Nachteile der Paarprogrammierung: Eine Untersuchung von Extreme Programming]. Addison-Wesley.

Cohn, M. (2009). *Succeeding with Agile: Software development using Scrum* [Erfolg mit Agile: Scrum in der Softwareentwicklung]. Pearson Education.

Consultancy. (2020, 7. Mai) *Half of companies applying Agile methodologies & practices [Die Hälfte aller Unternehmen verwenden agile Methoden und Praktiken]*. <https://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodologies-practices>

Cotton, I., & Greatorex, F. (1968). Data structures and techniques for remote computer graphics [Datenstrukturen und -techniken für Grafiken vom Remotecomputer] *AFIPS joint computer conferences: Herbst, Teil I* (533–544). Association for Computing Machinery. <https://doi.org/10.1145/1476589.1476661>

CruiseControl. (o. D.). *CruiseControl Homepage*. <http://cruisecontrol.sourceforge.net>

Date, C. J. (2019). *E. F. Codd and relational theory: A detailed review and analysis of Codd's major database writings* [E.F. Codd und die Theorie der relationalen Datenbanken: Überblick und Analyse seiner Hauptschriften]. Lulu Publishing Services.

Dastin, J. (2018, 11. Oktober). *Amazon scraps secret AI recruiting tool that showed bias against women* [Amazon verwirft geheimes KI-Tool zur Bewerbervorauswahl wegen Diskriminierung von Frauen]. Reuters. <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G>

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters [MapReduce: Vereinfachte Datenverarbeitung auf großen Clustern]. *OSDI '04: 6th symposium on operating systems design and implementation*, 6. USE-NIX Association.

- DevOps. (2015, 17. Februar). *Comparing DevOps to traditional IT: Eight key differences* [DevOps und traditionelles IT im Vergleich: Acht Hauptunterschiede]. DevOps. <https://devops.com/comparing-devops-traditional-eight-key-differences/>
- Docker. (o. D.-a). *Docker Homepage*. <https://docs.docker.com/>
- Docker. (o. D.-b). *Glossary* [Glossar]. <https://docs.docker.com/glossary/>
- Docker. (o. D.-c). *Docker hub*. <https://hub.docker.com/>
- Docker. (o. D.-d). *Docker swarm*. <https://docs.docker.com/engine/swarm>
- DonWells. (2013). Xp-feedback. [Feedback bei XP] Wikimedia Commons. <https://commons.wikimedia.org/wiki/File:XP-feedback.gif>
- Dudler, R. (o. D.). *Git: The simple guide* [Git: Der einfache Führer]. <https://rogerdudler.github.io/git-guide/>
- DVC. (o. D.). *Open-source version control system for machine learning projects* [Open-Source-Versionsverwaltungssystem für Machine Learning-Projekte]. Data Version Control. <https://dvc.org/>
- Dwork, C., Hardt, M., Pitassi, T., Reingold, O., & Zemel, R. (2012). *Fairness through awareness* [Fairness durch Kenntnis]. arXiv. <https://arxiv.org/abs/1104.3913v2>
- Eclipse. (o. D.). *Eclipse Homepage*. <https://www.eclipse.org>
- Elastic. (o. D.). *The ELK stack in a DevOps environment* [Der ELK-Stack in der DevOps-Umgebung]. <https://www.elastic.co/de/webinars/elk-stack-devops-environment>
- Fazliu, A. (2018, 2. Juni). *DevOps: To do or not to do? [DevOps: Einführen oder lieber nicht?]. Towards data science*. <https://towardsdatascience.com/devops-to-do-or-not-to-do-for-cus-on-culture-first-f82319ed346a>
- Fielding, R. (2000). *Architectural styles and the design of network-based software architectures* [Architekturtypen und das Design von netzwerkbasierter Softwarearchitektur]. Universität von Kalifornien.
- Fisher, R. A. (1936). *Iris data set* [Der Iris-Datensatz]. UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml/datasets/iris>
- Fowler, M. (2002). *Public vs. published interfaces* [Veröffentlichte im Vergleich zu öffentlichen Schnittstellen]. Martin Fowler. <https://martin-fowler.com/ieeeSoftware/published.pdf>
- Fowler, M. (2009). *TwoHardThings* [ZweiSchwereDinge]. Martin Fowler. <https://martinfowler.com/bliki/TwoHardThings.html>

Fowler, M., & Beck, K. (1999). *Refactoring: Improving the design of existing code* [Refactoring: Bestehenden Code verbessern]. Addison-Wesley.

Fowler, M., & Highsmith, J. (2001). The Agile Manifesto [Das Manifest für Agile Softwareentwicklung]. *Software Development*, 9(8), 28—35.

Free Software Directory. (o. D.). Subversion" /l "tab=Overview. <https://directory.fsf.org/wiki/Subversion%22/l%22tab%3DOverview>

Free Software Foundation. (o. D.-a) *GNU Wget [GNU Wget]*. GNU. <https://www.gnu.org/software/wget/>

Free Software Foundation. (o. D.-b). *GNU Gzip*. GNU. <https://www.gnu.org/software/gzip/>

Freeman, S., & Pryce, N. (2012). *Growing object-oriented software, guided by tests* [Objektorientierte Software und testgetriebene Entwicklung]. Addison-Wesley.

Gade, K. (2019, 13. Juni). *AI needs a new developer stack!* [Ein neuer Entwicklerstapel für künstliche Intelligenz]. Fiddler. <https://blog.fiddler.ai/2019/06/ai-needs-a-new-developer-stack/>

Git. (o. D.). *Git Homepage*. Git LFS. <https://git-scm.com/>

GitHub. (o. D.-a) *Github Homepage*. Git LFS. <https://docs.github.com/en>

GitHub. (o. D.-b). *Hello World* [Hallo Welt]. Git LFS. <https://guides.github.com/activities/hello-world/>

GitHub. (o. D.-c). *Git large file storage* [Git LFS]. Git LFS. <https://git-lfs.github.com/>

Google. (2020a). *Classification: Precision and recall* [Klassifikation: Genauigkeit und Abruf]. Google Cloud. <https://develop-ers.google.com/machine-learning/crash-course/classification/precision-and-recall>

Google. (2020b). *MLOps: Continuous delivery and automation pipelines in machine learning* [Kontinuierliche Lieferung und Automationspipelines beim Machine Learning]. Google Cloud. <https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>

GoCD. (o. D.). *GoCD Homepage*. <https://www.gocd.org>

Golden, B. (2014, 11. Februar). *How DevOps can accelerate the cloud application life-cycle* [Beschleunigung des Cloudanwendungslbenszyklus durch DevOps]. CIO. <https://www.cio.com/article/2378823/how-devops-can-accelerate-the-cloud-application-life-cycle.html>

- Hearn, S. (2019, 18. Januar). *5 great examples of agile organisations* [Fünf Musterbeispiele für agile Organisationen]. Clear Review. <https://www.clearreview.com/5-examples-of-agile-organisations/>
- Heck, P. (2020). *Testing machine learning applications* [Testen von Machine Learning-Anwendungen]. Fontys. <https://fontysblogt.nl/testing-machine-learning-applications/>
- Hermann, J., & Del Balso, M. (2017, 5. September). *Meet Michelangelo: Uber's machine learning platform* [Michelangelo: Die Machine Learning-Plattform von Uber]. Uber Engineering. <https://eng.uber.com/michelangelo-machine-learning-platform/>
- Highsmith, J. (2009). *Agile project management: Creating innovative products* [Agiles Projektmanagement und innovative Produkte]. Pearson Education.
- Humble, J., & Farley, D. (2015). *Continuous delivery: Reliable software releases through build, test, and deployment automation* [Kontinuierliche Lieferung: Zuverlässige Softwarrereleases durch die Automation der Build-, Test- und Bereitstellungsprozesse]. Addison-Wesley.
- H2O. (o. D.). *Explainable AI* [Erklärbare künstliche Intelligenz]. <https://www.h2o.ai/explainable-ai/>
- IBM. (o. D.-a) *AI Fairness 360* [Das IBM AI Fairness 360 Toolkit]. <http://aif360.mybluemix.net>
- IBM. (o. D.-b). *Rational ClearCase* [Rational ClearCase]. <https://www.ibm.com/products/rational-clearcase>
- IBM. (o. D.-c). *IBM Watson machine learning* [Machine Learning und IBM Watson]. <https://www.ibm.com/cloud/machine-learning>
- Import.io. (2017, 17. November). *Is web scraping legal? 6 misunderstandings about web scraping* [Ist Webscraping legitim? Sechs Missverständnisse]. <https://www.import.io/post/6-misunderstandings-about-web-scraping/>
- Jenkins. (o. D.-a) *Jenkins: Build great things at any scale* [Jenkins: Mächtig auf jeder Skala]. <https://www.jenkins.io/>
- Jenkins. (o. D.-b). *Build a Java app with Maven* [Java-Apps mit Maven]. <https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/>
- Jenkins. (o. D.-c). *Using a Jenkinsfile* [Wie verwendet man Jenkinsfile]. <https://www.jenkins.io/doc/book/pipeline/jenkins-file/>
- Jongerius, P., Offermans, A., Vanhoucke, A., Sanwikarja, P., & van Geel, J. (2013). *Get Agile! Scrum for UX, design & development* [Einstieg in Agile: Scrum für UX, Design und Entwicklung]. BIS Publishers.

Jupyter Team. (o. D.). *The Jupyter Notebook [Das Jupyter Notebook]*. <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>

Kaner, C. (2004). *A course in black box software testing: Examples of test oracles [Ein Kurs im Softwaretesten mit Black Box: Beispiele von Testrakeln]*. Florida Institute of Technology. <http://www.testingeducation.org/k04/OracleExamples.htm>

Karpathy, A. (2017, 11. November). *Software 2.0 [Software 2.0.]*. Medium. <https://medium.com/@karpa>

Katalon. (2020). *What is end-to-end (E2E) testing? All you need to know [End-to-End-Testen und alles was man dazu wissen muss]*. <https://www.katalon.com/resources-center/blog/end-to-end-e2e-testing/>

Kim, M., Zimmerman, T., DeLine, R., & Begel, A. (2018). Data scientists in software teams: State of the art and challenges [Datenwissenschaftler in Softwareteams: Der heutige Stand und die Herausforderungen]. *IEEE Transactions on Software Engineering*, 44(11), 1024—1038.

Kubeflow. (o. D.-a) *Kubeflow: An introduction to Kubeflow [Eine Einführung in Kubeflow]*. <https://www.kubeflow.org/docs/about/kubeflow/>

Kubeflow. (o. D.-b). *Kubeflow overview [Kubeflow im Überblick]*. <https://www.kubeflow.org/docs/started/kube-flow-overview/>

Kubernetes. (o. D.). *Kubernetes concepts [Wichtige Begriffe für Kubernetes]*. <https://kubernetes.io/docs/concepts/>

Kumar, R. (2016). *Human computer interaction [Mensch-Computer-Interaktion]*. Firewall Media.

Kwak, Y. (2005). A brief history of project management [Eine kurze Geschichte des Projektmanagements]. In E. G. Carayannis, F. T. Anbari, & Y. Kwak (Hrsg.), *The story of managing projects [Projekte managen - Eine Geschichte]* (S. 1—9). Praeger.

Lane, K. (2019, 10. Oktober). Intro to APIs: History of APIs [Einführung in APIs: Geschichte]. Postman. <https://blog.post-man.com/intro-to-apis-history-of-apis/>

Lumen. (2020) *Introduction to computer applications and concepts [Einführung in Computeranwendungen und Computerbegriffe]*. <https://courses.lumenlearning.com/computer-applications/chapter/reading-the-internet/>

Malamud, C. (1990). *Analyzing novell networks [Novell Netzwerke analysieren]*. Van Nostrand Reinhold.

Mangat, M. (2019, 4. November). *Kubernetes vs. Docker Swarm: What are the differences? [Kubernetes und Docker Swarm im Vergleich]*. PheonixNAP. <https://phoenixnap.com/blog/kubernetes-vs-docker-swarm>

- Masson, R., Iosif, L., MacKerron, G., & Fernie, J. (2007). Managing complexity in agile global fashion industry supply chains [Komplexität von agilen Lieferketten in der weltweiten Modebranche]. *The International Journal of Logistics Management*, 18(2), 238–254.
- Mehrjerdi, Y. Z. (2011). Six-Sigma: Methodology, tools and its future [Methodik, Werkzeuge und die Zukunft von Six Sigma]. *Assembly Automation*, 31(1), 79–88. <http://dx.doi.org/10.1108/01445151111104209>
- Mercurial. (o. D.). *Mercurial Homepage*. <https://www.mercurial-scm.org/>
- Misa, T., & Phillip, F. (2010). An interview with Edsger W. Dijkstra [Interview mit Edsger W. Dijkstra]. *Communications of the ACM*, Vol. 53(8). <https://doi.org/10.1145/1787234.1787249>
- Mitchell, I. (2015). Scrum Framework [Scrum Framework]. *Wikimedia Commons*. https://commons.wikimedia.org/wiki/File:Kanban_principles.jpg
- MLflow. (o. D.). *Built-in model flavors [Integrierte Modellvarianten]*. MLflow Project. <https://www.mlflow.org/docs/latest/models.html#built-in-model-flavors>
- MS Teams. (o. D.). *MS Teams Homepage*. Microsoft. <https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software>
- MS Visual Studio. (o. D.). *MS Teams Homepage*. Microsoft. <https://visualstudio.microsoft.com/vs/>
- Mundra, A., Misra, S., & Dhawale, C. A. (2013). Practical Scrum-Scrum team: Way to produce successful and quality software [Scrum-Scrum-Teams in der Praxis: Der Weg zu hochwertiger Software]. *13th International conference on computational science and its applications* (S. 119–123). IEEE Press.
- Newkirk, J., & Martin, R. C. (2000). Extreme programming in practice [Extreme Programming in der Praxis]. *OOPSLA '00: Addendum to the 2000 proceedings of the conference on object-oriented programming, systems, language, and application* (S. 25–26). Association for Computing Machinery.
- Nikitina, N., Kajko-Mattsson, M., & Stråle, M. (2012). From Scrum to Scrumban: A case study of a process transition [Der Weg von Scrum zu Scrumban: Eine Fallstudie]. *Proceedings of the international conference on software and system process* (S. 140–149). IEEE Press.
- North, D. (2006). Behavior modification, better software [Verhalten ändern, bessere Software produzieren]. *Stickyminds*, 2006(3). <https://www.stickyminds.com/better-software-magazine/behavior-modification>
- North, D. (2020). *Introducing BDD*[Einführung in verhaltensgetriebene Entwicklung]. Dan North & Associates. <https://dannorth.net/introducing-bdd/>

- Null, C. (o. D.). *Infrastructure as code: The engine at the heart of DevOps* [*Infrastructure as Code das Kernstück von DevOps*]. TechBeacon. <https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops>
- Pandas. (o. D.). *Pandas* [*Pandas*]. PyData. <https://pandas.pydata.org/>
- Patel, C. (2020, 2. Juli). *DevOps best practices* [*Best Practices in der DevOps-Kultur*]. DZone. <https://dzone.com/articles/devops-best-practices>
- Perforce. (o. D.). *Perforce Homepage*. <https://www.perforce.com>
- Price, D. R. (2005, 18. April). *CVS*. Wayback machine. https://web.archive.org/web/20120415051926/http://ximbiot.com/cvs/manual/cvs-1.12.12/cvs_1.html
- Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK Guide)* [*Der PMBOK-Guide*].
- Python Shell. (2020, 7. April). *Python-shell 1.0.4: Pip install python-shell* [*Python Shell 1.0.4: Pip install Python Shell*]. Python Shell Wrapper library. <https://pypi.org/project/python-shell/>
- Ranjan, P. (2014). PRINCE2 – Project management methodology [PRINCE2 – Projektmanagementmethodik]. Wikipedia Commons. https://en.wikipedia.org/wiki/PRINCE2#/media/File:PRINCE2_-_Project_Management_Methodology.png
- Ré, C., Niu, F., Gudapati, P., & Srisuwananukorn, C. (2019, September). *Overton: A data system for monitoring and improving machine-learned products* [*Overton: Ein Datensystem zur Überwachung und Verbesserung von Machine Learning-Produkten*]. Apple Machine Learning Research. <https://machinelearning.apple.com/research/overton>
- Red Hat Ansible. (o. D.-a). How Ansible works [Anleitung für Ansible]. Red Hat. <https://www.ansible.com/overview/how-ansible-works>
- Red Hat Ansible. (o. D.-b). *Automation: Learning Ansible basics* [*Automation: Die Grundlagen in Ansible*]. Red Hat. <https://www.redhat.com/en/topics/automation/learning-ansible-tutorial>
- Red Hat Ansible. (o. D.-c). What is container orchestration? [Container-Orchestrierung - Was ist das?]. Red Hat. <https://www.redhat.com/en/topics/containers/what-is-container-orchestration>
- Rimol, M. (2019, 10. Dezember). *The influence of digital business, cloud computing and hybrid IT creates significant ‘cascade effects’ that must be tackled by I&O leaders in 2020 and beyond* [*Der Einfluss von Digitalgeschäft, Cloud Computing und Hybrid-IT schafft einen signifikanten Dominoeffekt, der von Infrastruktur- und Betriebsfachleuten in Führungspositionen in den kommenden Jahren bewältigt werden muss.*]. Gartner. <https://www.gartner.com/smarterwithgartner/gartner-top-10-trends-impacting-infrastructure-operations-for-2020/>

- Rochkind, M. J. (1975). The source code control system [Das System zur Quellcodeverwaltung]. *IEEE Transactions on Software Engineering*, 1(4), 364–370.
- Rosenberg, D., & Stephens, M. (2008). *Extreme programming refactored: The case against XP[Extreme Programming unter der Lupe: Was spricht dagegen?]*. Apress.
- Rossberg, J., & Olausson, M. (2014). *Beginning application lifecycle management [Einstieg ins Application Lifecycle Management]*. Apress.
- Sadrach, P. (2020). *Why Keras is the leading deep learning API: Understanding the Keras deep learning library [Keras ist die führende Deep Learning-API: Ein Blick in die Keras-Bibliothek zeigt uns warum.]*. Towards data science. <https://towardsdatascience.com/why-keras-is-the-leading-deep-learning-api-31e00dfa012c>
- Sakolick, I. (2020). *What is CI/CD? Continuous integration and continuous delivery explained [Was bedeutet CI/CD? Einführung in die kontinuierliche Integration und Lieferung]*. Infowar. <https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html>
- Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems [Versteckte technische Schulden in Machine-Learning-Systemen]. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Hrsg.), *Advances in neural information processing systems 28 NIPS. NeurIPS Proceedings*. <https://papers.nips.cc/paper/2015/hash/86df7dcfd896fcf2674f757a2463eba-Abstract.html>
- Serrador, P., & Pinto, J. K. (2015). Does Agile work? A quantitative analysis of agile project success [Funktioniert Agile? Eine quantitative Analyse von erfolgreichen agilen Projekten]. *International Journal of Project Management*, 33(5), 1040–1051.
- Shaikh, A. (2017, 9. Juli). *An overview of IT roles, Waterfall and Agile [Überblick: IT-Rollen, Wasserfall und Agile]*. LinkedIn. <https://www.linkedin.com/pulse/department-ayub-shaikh/>
- Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile processes and methodologies: A conceptual study [Agile Prozesse und Methoden: Eine Studie der grundlegenden Begriffe]. *International Journal on Computer Science and Engineering*, 4(5), 892.
- Siegelau, J. (2004). How PRINCE2 can complement PMBOK and your PMK [PRINCE2 als Ergänzung zu PMBOK und PMK]. *Project Management Institute Global Congress 2004*.
- Sims, C., & Johnson, H. L. (2012). *Scrum: A breathtakingly brief and agile introduction [Scrum: kurz und agil]*. Dymaxicon.
- Speflow. (o. D.). *Gherkin*. <https://specflow.org/bdd/gherkin/>

- Spolsky, J. (2002, 11. November). *The law of leaky abstractions* [Das Gesetz der durchlässigen Abstraktionen]. Joel on Software. <https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/>
- Stallman, R. (2011). *The origin of the name POSIX* [Der Ursprung von POSIX]. Stallman. <http://stallman.org/articles/posix.html>
- StarTeam. (o. D.). *StarTeam Homepage*. Micro Focus. <https://www.microfocus.com/en-us/products/starteam/overview>
- Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and Kanban system materialization of just-in-time and respect-for-human system [Toyotas Produktionssystem und Kanbansystem verwirklichen ein Just-In-Time-System mit Respekt für den Menschen]. *The International Journal of Production Research*, 15(6), 553–564.
- Sutherland, J. (2004). Agile development: Lessons learned from the first Scrum [Agile Entwicklung: Erkenntnisse aus dem ersten Scrum]. *Cutter Agile Project Management Advisory Service: Executive Update*, 5, 1–4.
- Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017, Mai). Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages [Microservices in der agilen Softwareentwicklung: Workshopstudie der Schwierigkeiten, Vor- und Nachteilen]. *Proceedings of the XP2017 scientific workshops*, 23, 1–5.
- Takeuchi, H., & Nonaka, I. (1986). The new product development game [Das neue Produktentwicklungsspiel]. *Harvard Business Review*, 64(1), 137–146.
- Tensorflow. (o. D.). *Tensorflow Homepage*. <https://www.tensorflow.org>
- Terraform. (o. D.). *Terraform documentation* [Terraform-Dokumentation]. <https://www.terraform.io/docs/index.html>
- Thedev, J. (2019, 18. Juli). *The eight phases of a DevOps pipeline* [Die acht Phasen der DevOPs-Pipeline]. Medium. <https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba>
- Tichy, W. F. (1982). Design, implementation, and evaluation of a revision control system [Design, Implementierung und Beurteilung eines Revision Control Systems]. *Proceedings of the 6th international conference on software engineering* (S. 58– 67). IEEE Press.
- Tozzi, C. (2017, 9. Mai). *Understanding why Docker is so popular* [Warum ist Docker so beliebt? Eine Erklärung]. Container Journal. <https://containerjournal.com/features/understanding-why-docker-popular/>
- Travis CI. (o. D.). *Travis CI Homepage*. <https://travis-ci.org>

- Tsidulko, J. (2020). *Oracle v Google copyright case slated for Supreme Court arguments* [Der Urheberrechtsstreit zwischen Oracle und Google vor dem Obersten Gerichtshof der USA]. CRN. <https://www.crn.com/news/mobility/oracle-v-google-copyright-case-slated-for-supreme-court-arguments>
- Unadkat, J. (2020). *BDD vs. TDD vs. ATDD: Key differences* [Die wichtigsten Unterschiede zwischen BDD, TDD und ATDD]. Browserstack. <https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd>
- Universität von Maryland. (2018). *Resources for CMSC 341 Data Structures* [Ressourcen für SMSC 341-Datenstrukturen]. <https://user-pages.umbc.edu/~park/cs341.f18/resources/>
- Vargo, S., & Fong-Jones, L. (2018, 8. Mai). *SRE vs. DevOps: Competing standards or close friends? [SRE und DevOps: Ein Wettbewerb oder gute Zusammenarbeit?]*. Google. <https://cloud.google.com/blog/products/devops-sre>
- Verona, J. (2016). *Practical DevOps* [DevOps in der Praxis]. Packt Publishing.
- Vim. (o. D.). *Vim: the ubiquitous text editor* [Vim: Der Text-Editor ist überall.]. <https://www.vim.org>
- Visual Studio Code. (2020, 6. November). *IntelliSense* [IntelliSense]. <https://code.visualstudio.com/docs/editor/intellisense>
- VMware. (o. D.). *Container orchestration* [Container-Orchestrierung]. <https://www.vmware.com/topics/glossary/content/container-orchestration>
- Vorhies, W. (2017, 4. April). *DataOps – It's a secret* [DataOps - Ein Geheimnis]. Data Science Central. <https://www.datasciencecentral.com/profiles/blogs/dataops-it-s-a-secret>
- Waldner, J. (1992). Kanban principles [Kanban-Prinzipien]. Wikipedia Commons. https://commons.wikimedia.org/wiki/File:Kanban_principles.svg
- Wiggins, A. (2017). *The twelve-factor app* [Die Twelve-Factor-App]. 12 Factor. <https://12factor.net/>
- Wikipedia. (2020a). *Site reliability engineering*. https://en.wikipedia.org/wiki/Site_reliability_engineering
- Wikipedia. (2020b). *Transformational leadership*. [Transformationale Führung] https://en.wikipedia.org/wiki/Transformational_leadership
- Wikipedia. (2020c). *Data wrangling*. [Data Wrangling] https://en.wikipedia.org/wiki/Data_wrangling
- Wikipedia. (2020d). *Concept drift*. https://en.wikipedia.org/wiki/Concept_drift

Wikipedia. (2020e). *Data wrangling*. [Data Wrangling] https://en.wikipedia.org/wiki/Data_wrangling

Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). *Machine learning testing: Survey, landscapes and horizons* [Testen beim Machine Learning: Vergangenheit, Präsent und Zukunft]. IEEE Transactions on Software Engineering. <https://dx.doi.org/10.1109/TSE.2019.2962027>

Zliobaite, I. (2010). *Learning under concept drift: An overview*[Überblick über Machine Learning und Concept Drift]. arXiv. <https://arxiv.org/abs/1010.4784v1>

Z Shell. (o. D.). *An Introduction to the Z Shell* [Einführung in Z Shell]. SourceForge. http://zsh.sourceforge.net/Intro/intro_toc.html

ABBILDUNGSVERZEICHNIS

Abbildung 1: Wasserfall-Workflow	13
Abbildung 2: TPM-Workflow	14
Abbildung 3: PRINCE2-Struktur	18
Abbildung 4: Workflow im agilen Prozess	21
Abbildung 5: Kanban-Struktur	24
Abbildung 6: Online Kanban-Board	25
Abbildung 7: Kanban-Prinzipien	26
Abbildung 8: Kanban-Praktiken	26
Abbildung 9: Gedränge (engl. Scrum) beim Rugby	29
Abbildung 10: Agile Scrum Framework	30
Abbildung 11: Agile Scrum-Ereignisse	32
Abbildung 12: Extreme Programming	34
Abbildung 13: Prinzipien des Lean Management	37
Tabelle 1: Vergleich der Methoden des agilen und des klassischen Projektmanagements	38
Abbildung 14: DevOps-Pipeline	46
Abbildung 15: DevOps mit kontinuierlicher Integration und Bereitstellung	51
Abbildung 16: Monolithische Architektur im Vergleich zur Microservice-Architektur	55
Abbildung 17: Kontinuierliche Integration und Bereitstellung anhand von Git, Jenkins, Docker und Kubernetes	58
Abbildung 18: Docker-Komponenten	60
Abbildung 19: Docker-Image im Vergleich zu Docker-Container	60

Abbildung 20: Übersicht des Docker-Lebenszyklus	62
Abbildung 21: Kontinuierliche Integration und Bereitstellung anhand von Git, Jenkins, Ansible, Docker und Kubernetes	63
Abbildung 22: Kubernetes-Cluster	66
Abbildung 23: Kubernetes-Pods	67
Abbildung 24: Testquadrant nach Marick	73
Abbildung 25: Testautomationspyramide	75
Abbildung 26: Komponenten für ein ML-Modell	76
Abbildung 27: Lebenszyklus der testgetriebenen Entwicklung	79
Abbildung 28: TDD-Zyklus	80
Tabelle 2: TDD, BDD und ATDD im Vergleich	81
Abbildung 29: CD-Pipeline	84
Abbildung 30: Orchestrierte Experimente	86
Abbildung 31: CI/CD für ein ML-Modell	87
Abbildung 32: Lokales Versionsverwaltungssystem	89
Abbildung 33: Zentrales Versionsverwaltungssystem	90
Abbildung 34: Verteiltes Versionsverwaltungssystem	91
Abbildung 35: Momentaufnahmen in Git als Versionsverwaltung	92
Abbildung 36: Abschnitte in einem Git-Projekt	93
Abbildung 37: Lokale Repository-Bäume für Git	94
Abbildung 38: Verzweigung und GitHub	95
Abbildung 39: Syntaxhervorhebung in integrierter Entwicklungsumgebung	97
Abbildung 40: IDE-Komponenten in Eclipse	98
Abbildung 41: Landing Page von Jupyter Notebook	99

Abbildung 42: Ein Jupyter Notebook (.jpynb Datei)	100
Abbildung 43: Verwendung von Jupyter Notebook zur Ausführung eines einfachen Python-Codes	100
Abbildung 44: Verwendung von Markdown zur Dokumentation von Klartextcode in einem Jupyter Notebook	101
Abbildung 45: Grafische Darstellung von Daten in Jupyter Notebook	101
Abbildung 46: API-zentrische Ansicht	105
Abbildung 47: Die ersten Zeilen eines Datensatzes mit .head()	136
Abbildung 48: Übersicht über einen Datensatz mit .describe()	136
Abbildung 49: Modellentwicklung und Produktionslebenszyklus (I)	139
Abbildung 50: Modellentwicklung und Produktionslebenszyklus (II)	140
Abbildung 51: Modellentwicklung und Produktion mit Kubeflow	154
Abbildung 52: SageMaker Konsole	159
Abbildung 53: Zugriff auf IAM-Rollen	160

 **IU Internationale Hochschule GmbH**
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

 **Postanschrift**
Albert-Proeller-Straße 15-19
D-86675 Buchdorf

 media@iu.org
www.iu.org

 **Hilfe & Kontakt (FAQ)**
Antworten rund um Dein Studium findest
Du jederzeit auf myCampus.