KURSBUCH



## Softwareentwicklung für datenintensive Wissenschaften

DLMDSSEDIS01



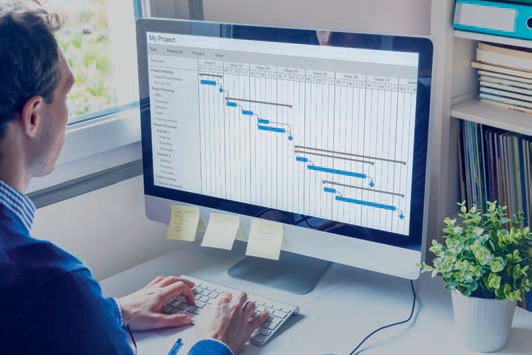
Lernziele

##### Einführung 9



Der Kurs „Softwareentwicklung für datenintensive Wissenschaften“ gibt Fachleuten in datenintensiven Wissenschaften, z.B. in Data Science, einen umfassenden Überblick über die Grundlagen der Softwareentwicklung. Lerneinheit 1 ist eine Einführung in zwei grundlegende Ansätze des Projektmanagements, nämlich die klassische Methode und die agile Methode. Beide Ansätze werden heute in Firmen praktiziert und haben Vor- und Nachteile. Ganz gleich welcher Ansatz am Arbeitsplatz gewählt wird, Projektmanagement ist die Grundlage produktiver Teamarbeit. Das Verständnis der Best Practices ist dabei hilfreich. In Lerneinheit 2 geht es um DevOps, d.h. die heute moderne Kombination aus Softwareentwicklung und -produktion innerhalb einer Firma. Die DevOps-Kultur ist die Grundlage für die Bereitstellung und Pflege von Dienstleistungen bei Unternehmen mit Vorreiterrolle, wie Google, aber auch bei den Startups der heutigen Zeit. Lerneinheit 2 bietet eine Einführung in die fundamentalen Bausteine skalierbarer Softwareumgebungen nach der DevOps-Philosophie. Das Wesentliche der Softwareentwicklung wird in Lerneinheit 3 behandelt. Sie beginnt mit dem Testen von selbstentwickelter Software und der Automatisierung von Tests in der Produktionspipeline. Des Weiteren werden das Thema Versionsverwaltung und die von Entwicklern in der Praxis verwendeten Werkzeuge abgedeckt. Anwendungsprogrammierschnittstellen, auch APIs genannt, bieten uns Interaktionsmöglichkeiten mit der Software und deren Diensten. Darum geht es in Lerneinheit 4. Wir werden Ihnen die wichtigsten Prinzipien zum Design und zur Erstellung von guten Benutzeroberflächen erklären, die Sie dann anwenden, um eine kleine Bibliothek in Python zu erstellen. In der letzten Lerneinheit werden die Unterschiede zwischen den Workflows von Datenwissenschaftlern und Softwareentwicklern, zwischen deren Anforderungen und Erwartungen sowie die sichere Integration und Umsetzung der Arbeit von Datenwissenschaftlern in skalierbare Produktionsumgebungen erläutert.

[www.iubh.de](http://www.iubh.de/)



# Lerneinheit 1

## Agiles Projektmanagement

#### LERNZIELE

Nach Abschluss dieser Lerneinheit wissen Sie...

...was klassisches Projektmanagement ist, welche Arten es gibt, und wie es definiert wird.

...was agiles Projektmanagement ist.

...was Kanban- und Scrum-Methoden sind.

...welche modernen Methoden im Projektmanagement gebraucht werden.

...wie ein Übergang vom klassischen zum agilen Projektmanagement gestalten werden sollte.

DL-E-DLMDSSEDIS01-U01

1. Agiles Projektmanagement

### Einführung

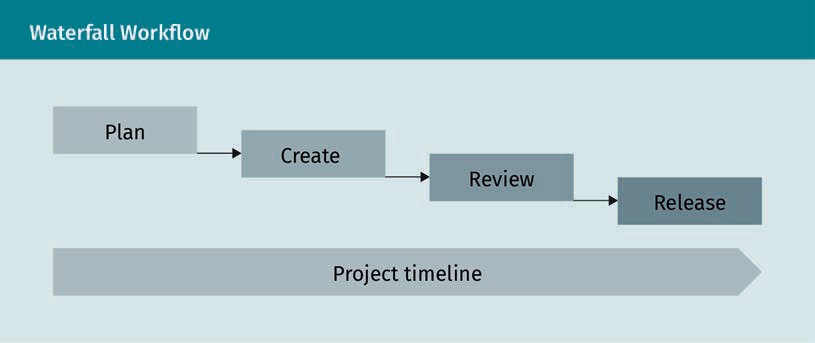
Die Arbeit einer Abteilung oder eines Teams kann in die Bereiche laufender Betrieb und neue Projekte aufgespalten werden. Im laufenden Betrieb werden tägliche Vorgänge abgedeckt. In einer Softwareentwicklungsabteilung eines Autoherstellers ist das Hauptziel beispielsweise das Erstellen der für das Unternehmen notwendigen Software. Die von der Betriebsabteilung normalerweise 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 dagegen werden Projekte, also spezielle, durch einmalige Ergebnisse und eindeutige Anfangs- und Endzeiten ausgezeichnete Aufgaben durchgeführt. Projekte sind keine langfristigen Ziele einer Abteilung, da sie generell temporär und von kurzer Dauer sind. Aus Projekten geht jedoch für ein Unternehmen überlebenswichtige Innovation hervor. Mit steigender Wichtigkeit von Innovationen werden auch Projekte und damit auch Projektmanagement und die entsprechenden Fähigkeiten immer wichtiger. Die Softwareentwicklungsabteilung eines Autohersteller könnte zum Beispiel mit einem Verkaufs- und Marketingprojekt beauftragt werden oder ein Entwicklungsprojekt für ein Autonavigationssystems bekommen, so dass das Unternehmen auf dem Markt konkurrenzfähig bleibt. Dazu braucht ein Unternehmen Projektmanagement, so dass 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 ja auch 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, die alle Aspekte des Projektmanagements abdecken, einzuführen. In den fünfziger Jahren des letzten Jahrhunderts begann die Industrie maßgeblich zu wachsen und Unternehmen begannen, systematisch Werkzeuge und Methoden des Projektmanagements in den Entwicklungsphasen ihrer Projekte einzusetzen (Kwak, 2005). Eine Leitlinie, die bis 2000 sehr intensiv zum Einsatz kam und die auch heute noch gebraucht wird, ist das klassische Projektmanagement (Project Management Institute, 2017).

### Klassisches Projektmanagement

WasserfallmodellEine lineare sequenzielle Entwicklungsmethode, bei der jede Phase bzw. jeder Schritt erst dann beginnt, wenn der vorherige Schritt (z.B. Softwaretesten) abgeschlossen ist.

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**:

Agiles Projektmanagement



Die Hauptidee des klassischen Projektmanagements ist die planbare und lineare Umsetzung von Projekten nach klar definierten und abgegrenzten Schritten mit dem Ziel, so dass ein Plan nicht stark geändert werden muss (Project Management Institute, 2017). In der heutigen Zeit haben manche Projekte klare Ziele und müssen während ihrer Laufzeit nicht oft angepasst werden. Dadurch können Ziele durch lineare, sequenzielle Ereigniskreise oder -phasen erreicht werden. Für solche Projekte eignet sich das klassische Projektmanagement, da dieser Ansatz einen festen Lebenszyklus mit folgenden Schritten hat.

Schritt nach dem Implementierungsschritt).



Nach Entstehen der Idee 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. Dann beginnt das Entwicklungsteam mit der Ausführung der im Projektplan definierten Aufgaben, während Status und Fortschritt jeder Phase vom Management überwacht wird. Nachdem alle Aktivitäten abgeschlossen und das Ziel erreicht ist, schließt der Manager das Projekt bzw. eine Phase formell ab. Beim klassischen Projektmanagement ist das Projekt normalerweise in klar definierte Phasen, die jeweils selbst Zielvorgaben und Ergebnisse haben, unterteilt. Bevor man von einer Phase in die nächste übergeht, müssen das Ziel der vorherigen erreicht sein, alle Aufgaben erfüllt und alle Entscheidungen getroffen werden. Eine besondere Betonung liegt beim klassischen Projektmanagement auf linearen Phasen, Vorausplanung, Dokumentation und der Priorisierung der definierten Aufgaben. Im gesamten Projektablauf 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. Und mit dieser Strategie ist das Ziel des klassischen Projektmanagements, die Effizienz des ursprünglichen Projektplans zu verbessern und dadurch die Ergebnisse im entsprechenden Zeitplan, Budget und Projektumfang zu erreichen.

Agiles Projektmanagement

Das klassische Projektmanagement hat folgende 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 remote ausgeführte Projekte in der Regel weniger Kommunikation, aber einen genaueren Plan erfordern. Beide Aspekte sind Stärken des klassischen Projektmanagements.
* Die Erreichung von Zielvorgaben zu geplanten Terminen führt zu erhöhter 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 Lerneinheit behandelt werden, findet weniger Austausch mit dem Kunden statt. Das kann zu Missverständnissen bezüglich des Ziels oder einzelner Ergebnisse des Projekts führen.
* Auch die Mitarbeiterkreativität 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.

Mehrere 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 Lerneinheit abgedeckt werden.

###### PMBOK

Project Management Body of Knowledge, kurz PMBOK, wurde vom Project Management Institute (PMI) eingeführt. Das 1969 gegründete PMI ist eine gemeinnützige Organisation, die Support anbietet und weltweit Standards für das Projektmanagement entwickelt (Project Management Institute, 2017). Das PMBOK wurde 1983 zum ersten Mal als eine Sammlung detaillierter Richtlinien und genormter und definierter Projektmanagementterminologie veröffentlicht. Es wird von Managern zur Standardisierung von Projekten in verschiedenen Teams und Organisationen gemäß PMI verwendet. Dem PMBOK zugrunde liegt die Annahme, dass Projektmanager einen Standard brauchen, 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 zusammen, 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 ist das Input für den nächsten. Jeder Prozess sollte ein bestimmtes Erfordernis des Projekts abdecken. Das Managementteam muss festlegen, welcher Prozess von wem, wie und wann verwendet wird.

Das PMBOK enthält auch Best Practices, Konventionen und Verfahrensweisen, 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). 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, so dass ein Projekt im Rahmen des genehmigten Budgets abgeschlossen werden kann.
5. Im Qualitätsmanagement werden Richtlinien festgelegt, mit Hilfe derer hochwertige Endergebnisse erzielt werden können.
6. Personalmanagement bedeutet hier die Organisation und die Führung von Projektteammitgliedern und das Management weiterer Betriebsmittel.
7. Zur Kommunikation gehören Erstellung, Sammlung, Verteilung, Verwaltung und Überwachung von Projektinformationen.
8. 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 Stakeholdermanagement müssen Menschen bzw. Organisationen, die Einfluss auf das Projekt haben könnten, identifiziert und 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 Best Practices nach den im PMBOK ausgeführten Prozessen erstellt werden. Ein Nachteil dieser Methode könnte es sein, dass die zahlreichen Kombinationen an Prozessen und Wissensbereichen für Projekte kleinen Umfangs eventuell zu kompliziert sind; daher müssen Umfang und Größe der Anwendung auf jedes Projekt angepasst werden.

Agiles Projektmanagement

###### PRINCE2

PRINCE2 steht für „PRojects IN Controlled Environments“ bzw. „Projekte in kontrollierten Umgebungen“ und wurde 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 wurde es zu einem bekannten Standard und wurde von Institutionen der Europäischen Union eingeführt (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 ein perspektiven- und produktbasiertes Projektmanagement-Modell. Der inhärente Produktfokus von PRINCE2 spiegelt die Realität und das Projektziel „Geschäftsbedarf als Projektergebnis“ wider. PRINCE2 stellt eine eingeständige Methode, deren Rahmenwerk auf sieben Prinzipien, sieben Themen und sieben Prozessen basiert, die Projektmanager anwenden und auf die Anforderungen jedes Projekts anpassen müssen (Siegelaub, 2004). Die folgenden Prinzipien sind die Grundprinzipien von PRINCE2:

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

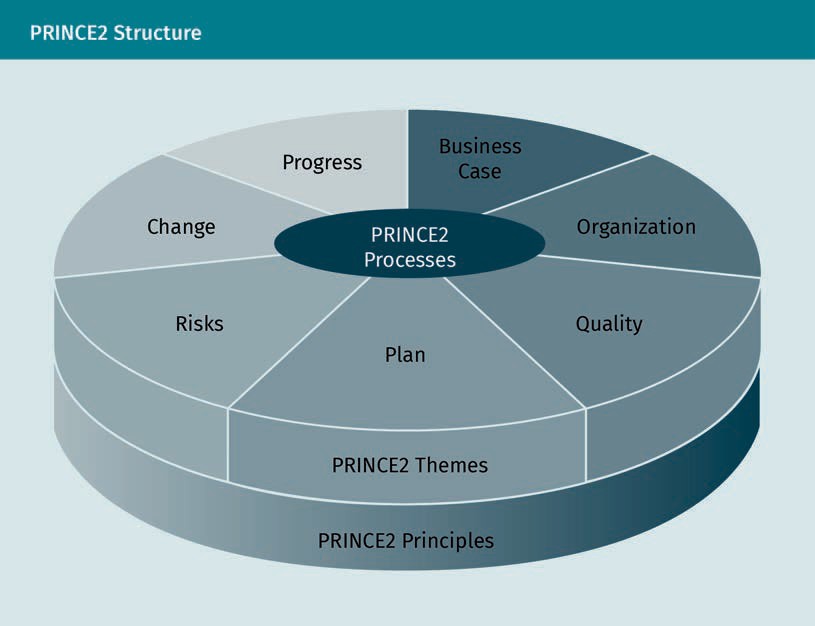
Im Lebenszyklus spielen auch sieben Themen bei PRINCE2 eine Rolle. Sie müssen ständig in die Projektaufgaben und -aktivitäten einfließen:

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 Zeitrahmen erreicht.
5. Fortschritt - Hierdurch wird der Fortschrift anhand von Status festgehalten, so dass das Team einen ständigen Überblick hat.
6. Qualität - Dieses Thema stellt den Bezug zum Prinzip Fokus auf Produkte her und gewährleistet Qualitätskontrolle zu jeder Projektphase.
7. Risiko - Dadurch werden die im Projekt möglichen Risiken identifiziert und gemanagt.

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

1. Vorbereiten eines Projekts
2. Initiierung eines Projekts
3. Lenken eines Projekts
4. Steuern einer Phase
5. Managen der Produktlieferung
6. Managen von Phasenübergängen
7. Abschließen eines Projekts

Die nachstehende Graphik zeigt die Struktur von PRINCE2.



Da es in den sieben Prozessen von PRINCE2 verschiedene Aktivitäten gibt, müssen die dafür jeweils Verantwortlichen und der Zeitpunkt ihrer Ausführung festgelegt werden (Bentley, 2010). Danach ist das Projektmanagement aufgrund der klaren Definition von PRINCE2 leichtes Spiel auch für Projektmanager mit wenig Erfahrung. Außerdem wird dadurch auch die Qualität des Projektergebnisses verbessert. Trotz des gut definierten 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.

Agiles Projektmanagement

### 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 Kundenaustausch. 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 Kunden oder Endverbraucher 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 ausgeführt wurden, eingeführt (Fowler & Highsmith, 2001). Die vier im Agilen Manifest festgelegten Werte lauten:

1. Menschen und Interaktionen stehen über Prozessen und Werkzeugen. Kommunikation und Zusammenarbeit mit dem Kunden 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. Zusammenarbeit mit dem Kunden steht über der Vertragsverhandlung. Die Unterzeichnung eines Vertrages mit dem Kunden ist weniger wichtig als die konkrete Zusammenarbeit. Ein Vertrag muss am Anfang sicherlich vorhanden sein. Mit der Zeit können sich Kundenanforderungen allerdings ändern, was dann gegebenenfalls 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, so dass der volle Projektumfang gewährleistet werden kann.

Wie bereits erwähnt, enthält das Manifest für Agile Softwareentwicklung auch 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 Kundenkommunikation und werden nachfolgend erläutert.

1. Kundenzufriedenheit: Sie wird durch frühe und kontinuierliche Lieferung eines Wertproduktes gewährleistet. Kunden sind zufriedener, wenn sie, anstatt auf den Release zu warten, regelmäßige Lieferungen funktionierender Software erhalten.
2. Anforderungsänderungen: Änderungen an den Anforderungen gehen ständig ein und werden während des gesamten Entwicklungsprozesses implementiert.
3. Lieferung: Funktionsfähige Software oder Produkte sollten, wenn möglich wöchentlich statt monatlich, an den Kunden geliefert werden.
4. Zusammenarbeit: Fachleute und Entwickler sollten während des Projekts eng zusammenarbeiten. Entscheidungen, über die sich Kunden bewusst sind und an denen sie auch teilhaben, sind in der Regel deutlich besser.
5. Mitarbeiterunterstützung: Menschen, die am Projekt mitarbeiten, sollten motiviert sein. Sie müssen unterstützt werden, und ihnen muss auch Vertrauen zukommen.

Agile

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

1. Kommunikation: Nach Auswahl des Entwicklerteams müssen persönliche Kommunikation und Zusammenarbeit unterstützt werden. Sie sind der Schlüssel zum Projekterfolg.
2. Fortschrittsmessung: Der Hauptmaßstab des Fortschritts ist funktionierende und funktionelle Software.
3. Nachhaltigkeit: Agile Prozesse führen zu nachhaltiger Entwicklung; Kunden, Entwicklern und Stakeholdern sollte es möglich sein, das Produkt auf Dauer weiterzuentwickeln.
4. Exzellenz: Es muss ständig Wert auf technisch exzellente Ausführung, Detail und Design gelegt werden.
5. Einfachheit: Das Projekt sollte einen klaren Zweck haben und klaren, allseits verständlichen Prinzipien folgen, so dass jeder eine Änderung vorschlagen kann.
6. Architektur: Die beste Architektur, gute Anforderungen und ausgezeichnetes Design kommen von selbstorganisierten Teams. Wenn Menschen motiviert sind, nehmen sie mehr Verantwortung an und treffen bessere Entscheidungen auch dadurch, dass sie mit anderen kommunizieren. Das führt zu außerordentlicher Zusammenarbeit und einem hochwertigen Produkt.
7. Feedback: Es sollte immer wieder Feedback zur Effektivitätsverbesserung angeboten werden. Denn Teams werden durch persönliche Weiterbildung sowie verbesserte Fähigkeiten, Werkzeuge und Prozesse effizienter.

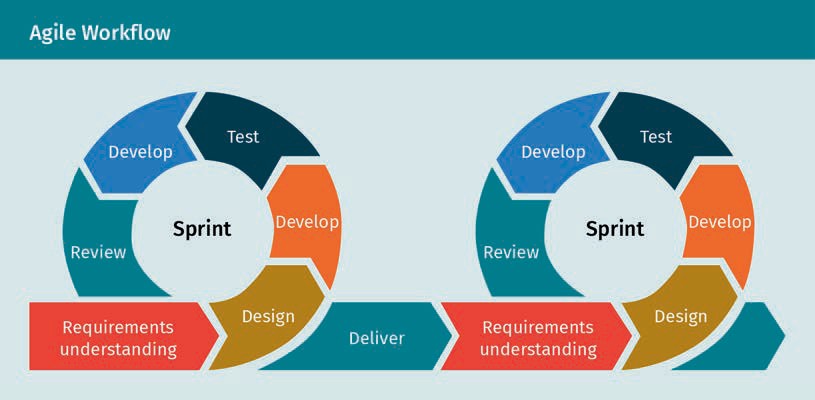
Nach Erläuterung der Werte und Prinzipien des agilen Projektmanagements können wir tiefer in diese Methode einsteigen. Agile ist eine iterative Form des Projektmanagements, die Organisationen dabei unterstützt, Produkte und Dienstleistungen schnell und problemlos an ihre Kunden zu liefern. Die Struktur 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 Zusammenarbeit einer Organisation gestaltet ist, die Zusammenarbeit und Leistung des Teams fördern. Agile beruht hauptsächlich auf Teamwork, Kommunikation, Zusammenarbeit, Timeboxing und schnelles Reagieren auf die Änderungen, die während des Projektlebenszyklus anfallen. Eine besondere Betonung liegt auch auf dem Software- und Projektentwicklungsprozess und vor allen Dingen auf der Qualität von Prozessen, Software und Projekten. 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 Kurzzyklus, 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 hat ein Team oder ein Entwickler einen bestimmten Zeitraum dauernden Iterationszyklus von einer bis sechs Wochen zur Abarbeitung einer Aufgabe innerhalb eines Moduls zur Verfügung.
* Optimierung. Alle unnötigen Aktivitäten werden bei der agilen Methode außen vor gelassen, um Risiken zu vermeiden und das Ziel zu erreichen.
* Anpassung. Die Anpassung an neue Risiken oder Änderungen findet beim agilen Projektmanagement schnell statt.
* Inkremente. Produkte müssen bei Agile in Inkrementen entwickelt werden, wodurch funktionelle Anwendungen in kleinen Schritten gebaut werden können. Zum Schluss werden die Inkremente in ein komplettes System oder Produkt integriert.
* Konvergenz. Im agilen Prozess fließen die Risiken der Inkremente zusammen.

Agiles Projektmanagement

* Zusammenarbeit. Der Agile Ansatz ist von Natur aus kommunikativ und kollaborativ.
* Kundenorientierung. Kundenzufriedenheit hat erste Priorität, das heißt Menschen stehen über den Prozessen oder der Technologie.

Die agile Methodik basiert auf einem iterativen Prozess, in dem Projekte in kleine Teile oder Inkremente, 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. Folgende Abbildung zeigt ein Beispiel dafür:



Im Großen und Ganzen heißt Agile weniger Zeit für die Planung oder die Priorisierung und mehr Zeit, um die Kundenanforderungen zu verstehen und Ergebnisse zu liefern. Da die Kundenzufriedenheit bei Agile oberste Priorität hat, sind Kunden bei allen Iterationen dabei und werden dazu aufgefordert, das gelieferte Produkt am Ende einer jeden Phase bzw. jeden Sprints zu überprüfen. Da Iterationen inhärent sind, ist diese Methode flexibler und kann daher leichter an die zur Kundenzufriedenheit notwendigen Änderungen angepasst werden können. Alle neuen Anforderungen werden normalerweise in der nächsten Schleife der Iteration berücksichtigt. Dadurch dass Projekte in kleine Abschnitte geteilt werden, hat ein Manager 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 bzw. Kunde: Der agile Prozess beginnt immer mit den Kunden und ihren Anforderungen.
* Produktinhaber: Der Produktinhaber (im Deutschen oft auch als Product Owner bezeichnet) verleiht den Kunden und internen Stakeholdern seine Stimme, das heißt ein Produktinhaber analysiert Ideen und fasst sie zusammen, sammelt nötiges Wissen und gibt Produktfeedback. Er arbeitet mit dem Entwicklungsteam, erstellt aus den Kundenanforderungen Benutzergeschichten und liefert zusätzliche Details. Diese sollten abdecken, wer der Benutzer oder Kunde ist, welche Art von Problem für sie gelöst wurde, warum die neue Lösung wichtig ist und welche Akzeptanzkriterien auf die Lösung anwendbar sind.

Sprint

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

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

Ein Produktinhaber ist ein Teammitglied, das die Verantwortung für die Formulierung der Vision und für die Erarbeitung einer daraus hervorgehenden, akzeptablen Lösung mit dem Entwicklungsteams 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 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 klar sein. Man sollte unter anderem folgende Vorteile für das Team und die Organisation beachten, 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 Kunden sind die Grundlage von Agile. Dadurch kann auch das Entwicklungsrisiko reduziert werden (Sharma et al., 2012).
* Vorzeitige Auslieferung eines Produkts ist gut für den Kunden (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).
* Wenn Änderungen angenommen und Kunden bei jedem Schritt bzw. jeder Entscheidung miteinbezogen werden, steigt die Qualität des Endprodukts (Masson et al., 2007).

Agile ist aber nicht perfekt und der Erfolg des Projekts ist dadurch nicht garantiert. Daher sollten auch folgende 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 Entwicklern eventuell weniger Informationen zur Verfügung stehen (Sharma et al., 2012).
* Die Verbesserung eines Produkts beruht beim agilen Prozess auf klarem Feedback vom Kunden und auf dessen guter Kommunikation. Wenn Kunden 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 Entwicklern bestehen, die schnell arbeiten können und wenig Zeit zur Projektplanung brauchen. Dadurch entstehen höhere Kosten (Taibi et al., 2017).

Agiles Projektmanagement

Trotz dieser Nachteile ist Agile eine erfolgreiche Methode, die in der IT-Branche zu qualitativ hochwertigen Produkten 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). 2012 hat beispielsweise Adobe Agile mit dem Ziel eingeführt, den Managementaufwand zu reduzieren und konnte 80.000 Stunden pro Jahr dadurch sparen (Hearn, 2019). Im Lauf der Zeit wurden verschiedene, agile Ansätze in verschiedenen Branchen zur Lieferung erfolgreicher Systeme eingeführt, entwickelt, getestet und freigegeben Zu den bekanntesten gehören Scrum und Kanban.

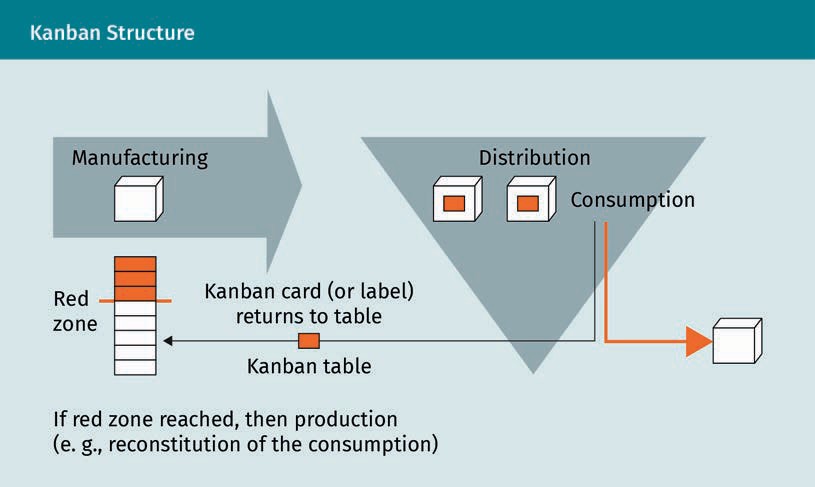
### Kanban

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 Kundenbedarf geliefert. Der aktuelle Marktbedarf oder Kundenanfragen blieben unberücksichtigt. Dagegen ist Kanban ein Pull-System und Teil der agilen Bewegung. Ein Pull-System basiert auf tatsächlichen Kundenanfragen und schickt eine Aufgabe erst dann an die Fertigungslinie, wenn Bedarf für ein bestimmtes Produkt besteht. Kanban wurde für den Einsatz in verschiedenen Industrien modifiziert, was zu weniger Aktivität bei gleich bleibender Produktivität geführt hat. Gleichzeitig stieg bei gleichen Kosten für den Betrieb der Mehrwert für den Kunden. Im Mittelpunkt dieser Methode stehen Aufgaben; ihre Ausführung, der Zeitpunkt der Ausführung und die Menge werden visuell angezeigt. Drei Teile sind dabei wichtig: das Kanban-Board, Kanban-Karten und Swimlanes.

Folgende Abbildung zeigt das ursprüngliche Kanban-System:

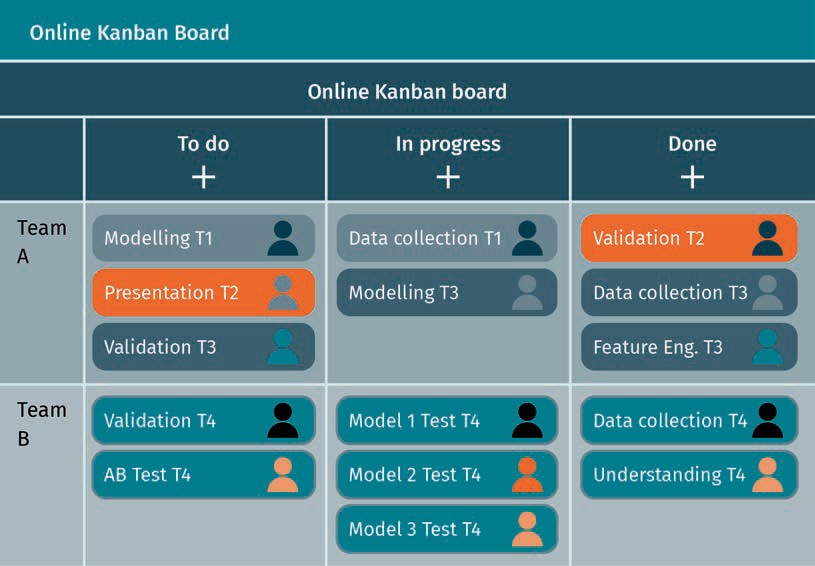
Pull-System

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



Eine Kanban-Karte wird im Kanban-Workflow auf ein Verbrauchssignal aus der roten Zone entnommen. Gleichzeitig wird in der Fertigungslinie eine Aufgabe gestartet. Nach 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 die Aufgabenstatus anzeigen, und deren Anzahl je nach Team- und Projektaufbau angepasst werden. Drei Spalten sind jedoch immer auf dem Board: eine für Projektideen, ein für laufende Arbeiten, und eine für abgeschlossene Arbeiten. Sie könnten zum Beispiel mit „Offen“, „Laufend“ und „Erledigt“ 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 eine Kanban-Board.

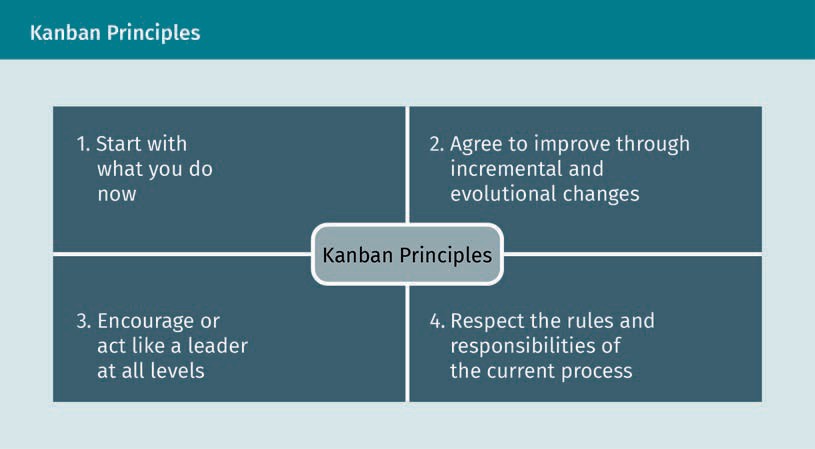
Agiles Projektmanagement



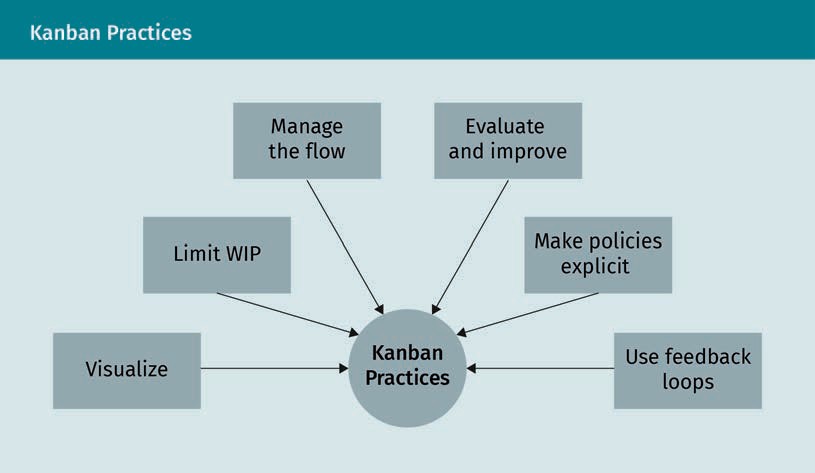
Aufgrund des technischen Fortschritts und des wachsenden Einsatzes des Internets bieten viele Online-Plattformen heute Kanban an. Dennoch folgen viele Firmen der traditionellen Kanban-Philosophie, bei der tatsächliche Tafeln bevorzugt werden, da sie den Austausch fördern (Collado, 2018). Durch das Kanban-Board 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, einführt (Sugimori et al., 1977). Die vier Prinzipien werden in der folgenden Abbildung dargestellt:



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 berücksichtig werden sollten und die in der nachstehende Abbildung veranschaulicht werden:



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, Besitzer 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, so dass die Produktivität und die Konzentration gewährleistet werden kann. Jede Aufgabe wird innerhalb eines Sprints ausgeführt. Falls das im entsprechenden Inkrement nicht möglich ist, kann sie ins nächste Inkrement übernommen werden. Bei Einführung der Kanban-Methode ist es nicht einfach, das Limit an „in Arbeit“ befindlichen Aufgaben einzuhalten.

Agiles Projektmanagement

Daher empfiehlt es sich, niedrige bzw. angemessene Limits anzusetzen, 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, so dass klar bleibt, wer wann was macht. 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 Menschen bzw. Arbeit zu verbinden. Ein interessanter Aspekt bei Kanban ist das Feedback, 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 dem Aufgabenbesitzer Feedback zu vermitteln.

###### Vorteile von Kanban

Zur Einführung von Kanban sind keine großen Veränderungen notwendig, da Begriffe, Implementierung und Verfahren auch für Mitarbeiter 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 fürs Team, 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 im 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 das Team sie schnell integrieren.

###### Agiles Kanban

Agiles Kanban ist eine Kombination aus den Prinzipien der agilen Projektmanagementmethodik 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 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. Der Teamleiter sollte das Limit der „in Arbeit“ befindlichen Aufgaben der Fähigkeit des Teams entsprechend ansetzen, so dass während des Entwicklungszyklus produktiv, konzentriert und flexibel gearbeitet werden kann. Die Aufgaben werden nach Kundenanforderungen 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 den Kunden. 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 eventuell besser für Organisationen, die mit einem Push-System and komplexeren Aufgaben und mit vielen Stakeholdern arbeiten. Händler arbeiten beispielsweise mit Push-Systemen. Ein neues Produkt wird anhand des Kundenbedarfs für andere Produkte auf dem Markt eingeführt und dann durch entsprechendes Marketing gefördert.

### 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:

Agiles Projektmanagement



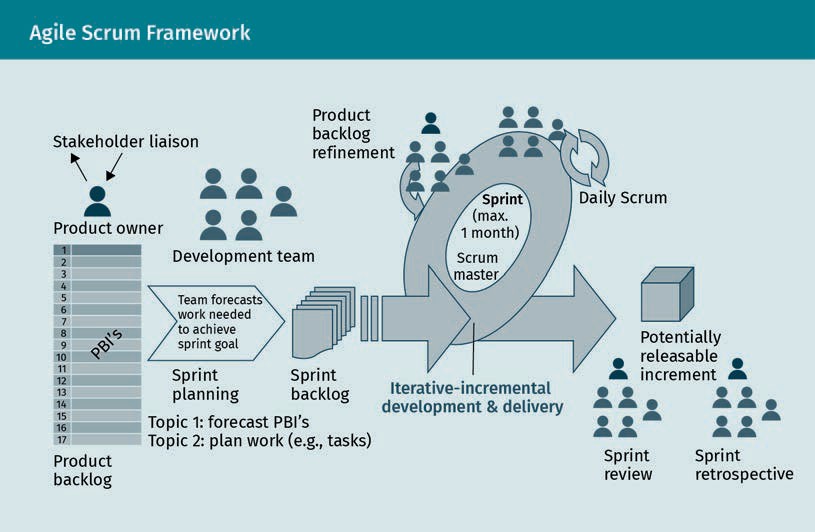
Die Methode des Agile Scrum wurde von Schwaber und Babatunde am DuPont Forschungszentrum und der Universität von Delaware in der Softwareentwicklung eingeführt. Sie wurde zunächst für die Teamführung verwendet und dann in der Easel Software Development Corporation angewandt (Sutherland, 2004). Entwicklungsteams 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 regelmäßige Kommunikation miteinander und mit anderen Stakeholdern verbessern sowie Problem beim ihrer Entstehung gemeinsam abarbeiten. Dieses Modell eignet sich am besten für Firmen oder Teams, deren Produktentwicklungsprozess in zwei- bis vierwöchige Iterationen unterteilt werden kann.

###### Agile Scrum Framework

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

* + - Transparenz: Die Arbeitsumgebung und -kultur sollte den Austausch von Wissen über den Prozess und die inhärenten Hindernisse ermöglichen, so dass 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 keinen Projektmehrwert beitragen, eliminiert werden können.

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



###### 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 Kundenprüfung steht. Die zum Abschluss eines Sprints notwendige Zeit sollte gleich bleiben, so dass Kosten genau prognostiziert werden können. Im Idealfall dauert ein Spring ein bis zwei Wochen. Mit einem zweiwöchigen Sprint betragen die Kosten für die Mitarbeiter beispielsweise die Hälfte der Monatsgehälter.

Agiles Projektmanagement

###### Rollen und Agile Scrum

Beim 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 Produktinhabers, die praktisch dieselbe ist, wie die des Produktinhabers bei Agile.

* + Produktinhaber sind für die Steuerung des Produktbacklogs verantwortlich, so dass das erwünschte Ergebnis und die Kundenzufriedenheit 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, so dass alle Anforderungen zur erfolgreichen Ausführung der Aufgaben erfüllt sind. Während des Sprints vertreten Scrum Master das Team und arbeiten eng mit den Produktinhabern.
  + Die Rolle des Entwicklers eignet sich hervorragend für passionierte Produktentwickler im Team, die die notwendigen Fähigkeiten zur Lieferung eines qualitativ hochwertigen Produkts und zur Kundenzufriedenheit mitbringen.

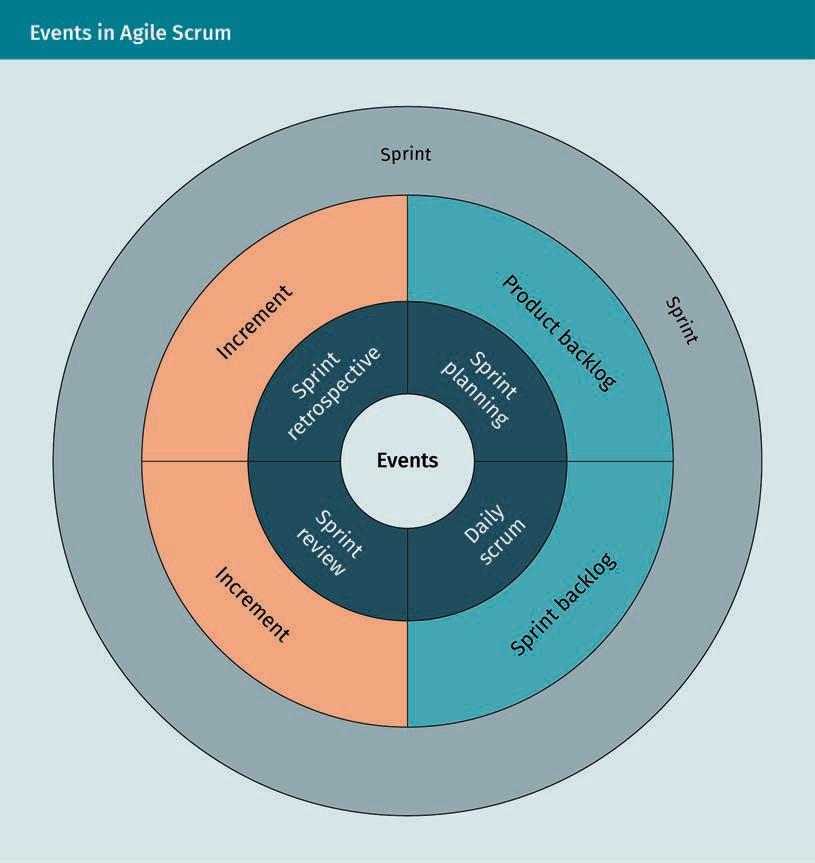
###### Artifacts als essenzielle Elemente des Agile Scrum

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

* + Der Produktbacklog ist eine sortierte Liste von Anforderungen, die zur Umsetzung von Projektänderungen gebraucht werden. Die einzelnen Artikel in einem Produktbacklog werden als Benutzergeschichten bezeichnet. Die Pflege des Produktbacklogs unterliegt dem Produktinhaber, der auch den Geschäftswert der einzelnen Artikel bestimmt. Artikel in einem Produktbacklog sind Optionen, die eine Entwicklungsteam wählen und im nächsten Sprint abarbeiten kann. Das heißt allerdings nicht, dass alle Artikel in einem Produktbacklog geliefert werden.
  + Der Sprintbacklog 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 einen Inhaber. Dieser Inhaber 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 Produktinhaber dem Sprintbacklog 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 Kunden geliefert werden.
  + Die „Definition von Erledigt“ wird vorher vom Team anhand von Standards und Regeln festgelegt. Diese werden verwendet, wenn Artikel zum Sprintbacklog 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 Artifacts in Agile werden in folgender Abbildung dargestellt:



* + - Sprintplanung: Dabei dreht es sich um ein Besprechung am Anfang jeden 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 Produktbacklog auswählen und in den Sprintbacklog übernehmen.

Agiles Projektmanagement

Nach diesem Schritt entscheidet der Produktinhaber mit dem Team, welche Artikel ausgewählt werden. Das Entwicklungsteam bestimmt daraufhin die Akzeptanzkriterien für das Ergebnis und die Lieferung des Produkts.

* + Sprintreview: Am Ende des Sprints geht das gesamte Team die Aufgaben noch einmal durch, die im letzten Sprintbacklog ausgeführt wurden. Stakeholder und Kunden, 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 dann dem Backlog zukünftiger Sprints als neue Artikel hinzugefügt werden.
  + Sprint-Retrospektive: Am Ende eines Sprintreviews können das Entwicklerteam und der Produktinhaber den Sprint, die guten und die schlechten Element 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 zur mehr Produktivität führt.

###### Vor- und Nachteile von Agile Scrum

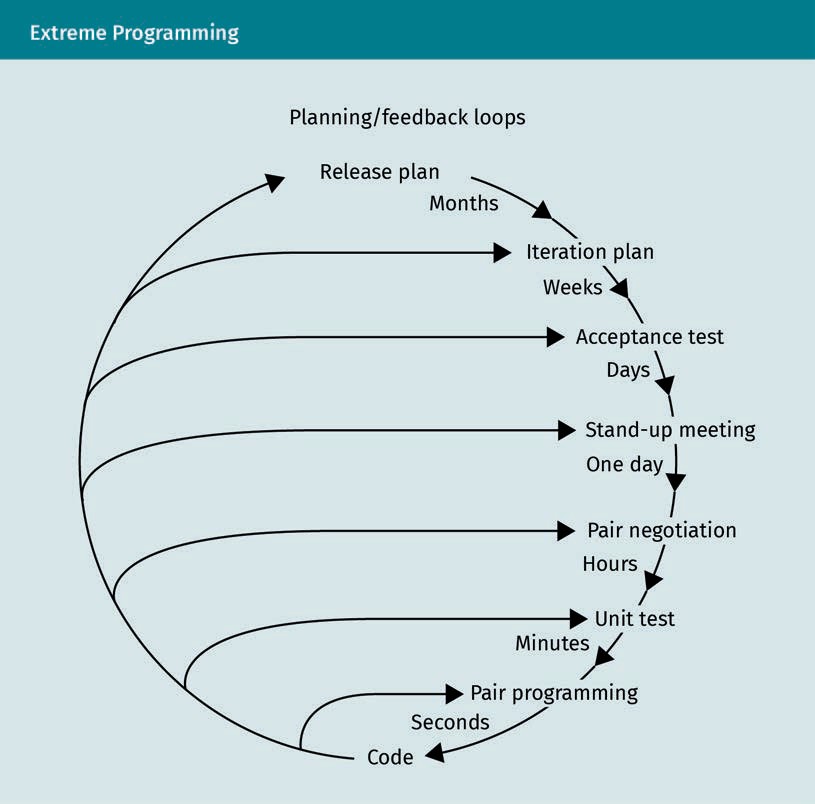
Agile Scrum eignet sich besonders für Branchen mit komplexen Projektprozessen, die Ergebnisse bewerten müssen und die um Kundenzufriedenheit bemüht sind (Jongerius et al., 2013). Am meisten hat die Softwarebranche und vor allen Dingen kleine Projektteams von maximal zehn Leuten davon profitiert. Agile 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 machen, wenn besondere Fertigkeiten verlangt werden, oder wenn viele Stakeholder oder externe Abhängigkeiten vorhanden sind. Unter obigen Voraussetzungen sollte ein Team dennoch überlegen, ob es Agile Scrum eingesetzten möchte, denn diese Methode kann immer an die individuellen Anforderungen eines Teams angepasst, womit sich diese Nachteile möglicherweise ausschalten lassen.

### Andere moderne Methoden

Seit Jahrzehnten ist durch klassische und agile Managementmodelle eine Grundlage für die Projektmanagementbedürfnisse von Organisationen entstanden. Wenn diese Methoden über den Rahmen eines einzelnen Teams hinaus eingesetzt werden sollen, müssen die Ressourcen einer Firma, die Anforderungen und die Fähigkeiten in Betracht gezogen werden, so dass Projekte abgeschlossen und damit gute Resultate erzielt werden können. Da 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, brauchen Teams einen guten Überblick über die verschiedenen Methodiken 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 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 Anforderung ändern und ständiges Feedback notwendig ist. Das XP-Modell beruht auf vier Grundaktivitäten: umfangreiches Codieren, Modultesten, Kunden 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:



Daraus ist ersichtlich, dass Kunden 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).

Agiles Projektmanagement

###### Adaptive Project Framework (APF-Methode)

Klassische Projektmanagementmethoden lassen sich nicht leicht an Veränderungen und Kundenfeedback anpassen. Dazu muss eine Organisation die Projektmanagementmethodik prozess- und phasenübergreifend anhand eines soliden Projektplans 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 eine Entwicklungsteam in Kommunikationszyklen statt nach Sprints arbeitet. Kunden und Stakeholder können in jedem Zyklus Änderungen am Projektumfang vorbringen, die ein Entwickler überdenkt und gegebenenfalls in der neuen Phase implementiert. Obwohl Kunden und Stakeholder bei APF 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 der Geschäftsprozesse mit Hilfe dieser Methode zu verbessern. Sie wurde von einem Ingenieur bei Motorola entwickelt, der datengetriebene Methodiken im Projektmanagement einführen wollte (Mehrjerdi, 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 bekommen. Durch die Inputsteuerung können Organisationen ein Projekt erfolgreich liefern, denn dadurch wird der Prozess vorhersehbar und Ursachen von während der Projektentwicklung auftretender Fehler können vermieden werden. Die Prozessvariationen werden also minimiert und Ergebnisse können zuverlässig und zu niedrigeren Kosten erzielt werden. Von der Leitung bis zum Entwickler müssen bei dieser Methode jedoch Strukturen, Einsatz und Fähigkeiten vorhanden sein. Eine Wissenslücke oder fehlende Verantwortung kann große Auswirkungen auf das Output haben. Da Steuerung ein wichtiger Teil dieser Methode ist, hat sie den Nachteil, dass bei Benutzern schnell das Gefühl des Micromanagements aufkommen kann.

###### Scrumban

Mit der Weiterentwicklung von Kanban entstand Scrumban, um Scrum-Teams die Vorteile von Kanban zu ermöglichen (Nikitina et al., 2012). Die Struktur ist ähnlich wie bei Scrum, die Arbeitskultur entspricht die von Kanban. Bei Scrumban wird ein Kanban-Board verwendet und kleine Iterationen, in Anlehnung an den Scrum Sprint, organisiert. 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, so dass 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 mehr Karten auf dem Kanban-Board sind. Es wird also sichtbar gemacht, was in Arbeit ist, um die Produktivität des Teams zu erhöhen.

Daher ist diese Methode besonders für Organisationen geeignet, 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, verschiedene Geschäftsbereiche mit mehreren Produktlinien zu koordinieren und die Verbindung zwischen Teams zur Lieferung komplexer Lösungen herzustellen. Es wird in verschiedenen Organisationen und Branchen, z.B. der Software-Branche, verwendet (Mundra et al., 2013). Durch diese Methode werden leistungsstarke Scrum-Teams zusammengeführt, um an den Projektzielen zu arbeiten. Dabei entsteht unter den Teammitgliedern Respekt und Vertrauen und sie sind auf die gemeinsamen Projektprozesse eingestellt. Ideal dafür sind daher Teams mit maximal sechs Mitgliedern, denn größere Teams können die Kommunikation und Produktbereitstellung besonders erschweren, während die Aufteilung in kleinere Teams gute Strukturen schafft und zu besseren Resultaten führt. Bei der durch Scrum of Scrum geschaffenen Struktur wird jedes Team durch den Produktinhaber mit eine Kernaktivität, der sogenannten Organisationslieferung, verbunden. Produktinhaber sind deshalb dafür verantwortlich, dass Informationen zu Bedarf und Umfang zwischen den Teams fließen. Der Hauptproduktinhaber managt das die Zusammenarbeit der Produktinhaber und führt sie anhand der Vision für das Produkt (Mundra et al., 2013). Produktinhaber nehmen täglich an Stand-Up-Meetings teil, in denen die Sprintziele, Hindernisse und Verbesserungen bezüglich anderer Projekt, für die ihr Team verantwortlich ist, besprochen werden.

###### Lean Management

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 höherer Leistung und steigenden Werten (Ballard & Howell, 2003). Das Modell beruht auf drei Hauptregeln:

* + - Mehrwert für den Kunden schaffen.
    - Verschwendung oder Prozesse ohne Mehrwert für das Endprodukt oder Vorteil für den Kunden abschaffen.
    - Produkte und Prozesse kontinuierlich verbessern.

Bei dieser Methode geht es nicht um den Projektentwicklungsprozess, sondern um den Kunden, die Verbesserung der Arbeitsprozesse und die gesteckten Ziele. Dabei sollte Verantwortung unter den Teammitgliedern geteilt werden, und Teamleiter sollten auch einmal Führungsaufgaben übertragen. Aufgrund dieser Methode konnte Prozesse abgeschafft oder gekürzt und Kosten gesenkt werden. 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.

Agiles Projektmanagement



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

### Einführung von Agile

Das schnelle Wachstum des Technologiesektors und des Markts kann sich bei Firmen in Infrastrukturänderungen auswirken. 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 Automobilehersteller 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 notwendig und man muss mit Risiken aufgrund der hohen Änderungsrate rechnen. Wenn eine Firma jedoch auf dem Markt bestehen will, muss sie unter Umständen jedoch auf Agile umstellen und ein passendes Modell finden.

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

|  |  |  |
| --- | --- | --- |
| Vergleich der Methoden des agilen und des klassischen Projektmanagements | | |
|  | Klassisches Projektmanagement | Agiles Projektmanagement |
| Anforderungen | Klare Anforderungen und wenige Änderungen | Klare Anforderungen und viele Änderungen |
| Kunde | 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 bedenken |

Wenn Projekt nicht termingerecht, mit entsprechendem Standard und unter Erfüllung der Akzeptanzkriterien des Teams oder des Kunden geliefert werden können, sollte untersucht werden, ob der Projektmanagementansatz nicht für diese Mängel verantwortlich ist. Klassisches Projektmanagement kann unnötig komplex und uneffektiv sein. Die Einführung eines neuen, verbesserten Systems, wie Agile, könnte dem Team wieder zu Erfolg verhelfen. Für Teams im Bereich von Datenanalyse oder Data Science könnte eine Mischung aus verschiedenen agilen Ansätzen, z.B. Scrumban, das beste Modell sein. Kommt diese Methodik zum Einsatz, kann das Team Änderungen, direkt nachdem vom Kunden Feedback eingegangen ist, umsetzen. Zu Änderungen ist jedoch wahrscheinlich trotzdem Zeit notwendig, da vielleicht mehr Daten notwendig sind oder der Kunde den richtigen Augenblick abwarten muss, um Wissen einzubringen. In der Softwareentwicklung können Änderungen dagegen relativ leicht umgesetzt werden, da selbst ohne Kundeninput 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 den Kunden passendes Modell notwendig ist. Eine plötzliche Umstellung vom klassischen auf agiles Projektmanagement kann ein Team besonders fordern, zumal wenn die Teammitglieder nicht mit der Methodik vertraut sind und daher einen völlig neuen Ansatz lernen 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

Agiles Projektmanagement

In eine neue Sprach und Arbeitskultur angepasst werden. Insgesamt gesehen bedarf ein reibungsloser Übergang von einem alten auf eine neues Projektmanagementsystem, wie Agile, dass viele neue Fähigkeiten erlernt werden und geduldig vorgegangen wird. Trotz dieses Aufwands lohnt sich der Versuch.

Zusammenfassung

Diese Lerneinheit 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 Kundenzufriedenheit durch Erreichung der Projektziele. Diese Methode ist jedoch linear und die Terminplanung ist nicht einfach. Kundenänderungen können nicht integriert werden und die Kreativität der Teammitglieder wird durch den festen Zeitplan gehemmt. Als zweites wurde das Framework des agilen Projektmanagements, die Rollen, Prinzipien und Eigenschaften abgedeckt. Agile hat kürzere Entwicklungsphasen, benötigt zu Projektbeginn weniger Details, beruht aber auf effektiver Kommunikation zwischen Entwicklern und Kunden und führt zu höherer Produktivität des Entwicklungsteams. Danach wurde Kanban erörtert. Bei dieser Methode kommen Kanban-Boards zur Visualisierung der Aufgaben zum Einsatz. Kanban hat einen einfachen Workflow, bei dem Änderungen und Kundenwünsche akzeptiert werden, wodurch das Entwicklungsteam viel Zeit spart. Eine weitere Methode des agilen Projektmanagement, die besprochen wurde, ist Scrum. Scrum basiert auf einem iterativen Entwicklungsprozess, dem Sprint, bei dem alle Aufgaben zwischen drei Hauptrollen verteilt werden: Produktinhaber, Scrum Master und Entwickler. Ein Team kann durch Scrum produktiver arbeiten und Kundenänderungen am Ende eines Sprints integrieren. Auch Philosophie, 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 Kundenzufriedenheit aufgrund der möglichen Integration von Änderungswünschen des Kunden während der iterativen Entwicklungsphase.



# Lerneinheit 2

## DevOps

#### LERNZIELE

Nach Abschluss dieser Lerneinheit verstehen Sie ...

... die Begriffe DevOps und Lifecycle Management.

... den Unterschied zwischen DevOps und klassischem Lifecycle Management.

... die Ziele von DevOps.

... die Auswirkungen von DevOps auf Teams und die Entwicklungsstruktur.

... wie eine DevOps-Infrastruktur entsteht.

... wie skalierbare Umgebungen entstehen.

DL-E-DLMDSSEDIS01-U02

1. DevOps

### Einführung

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

Die Benennung DevOps ist eine Kombination der 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 haben das gemeinsame Ziel, von der Konzeption bis zur Auslieferung an den Kunden oder Benutzer stabile und hochwertige Software so schnell wie möglich zu implementieren (Amazon, o.D.-a). DevOps ist weder Werkzeug noch Software, Technologie, Methodik oder Prozess. DevOps ist eine Unternehmenskultur mit Prinzipien, die ein Unternehmen langfristig umsetzen möchte.

Bevor wir tiefer in die Thematik 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 der Idee zur ihrer Implementierung und dann zu einer Lösung beschrieben. Durch iterative Zyklen werden Mitarbeiterzusammenarbeit, Rollen, Prozesse und Information kontinuierlich verbessert. Im Application Lifecycle Management werden mehrere Disziplinen, die im Rahmen früher angewendeter Entwicklungsprozesse oft separat behandelt wurden, zusammengefasst: Projektmanagement, Anforderungsmanagement, Softwareentwicklung, Testen, Qualitätssicherung, Bereitstellung, Wartung usw. DevOps wird durch das Application Lifecycle Management unterstützt, indem alle 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.

### Klassisches Lifecycle Management

Beim klassischen Lifecycle Management werden die Rollen in einer Organisation getrennt voneinander behandelt (Shaikh, 2017). Entwickler kommen mit Änderungen gut klar, da Änderung ständiger Bestandteil ihrer Aufgabe ist. Um auf anspruchsvolle Änderungen eingehen zu können, benötigt eine Firma Entwickler, die Innovation und Änderung anstreben. Im Betrieb sind Änderungen eher hinderlich. Ein Unternehmen braucht eine Betriebsabteilung zur reibungslosen Abwicklung von Abläufen und Lieferung von gewinnbringenden Leistungen. Betriebsabteilungen lehnen Änderung zu Recht ab, da dadurch Stabilität und Betriebssicherheit des Unternehmens gefährdet wird.

DevOps

###### Entwickler

Entwickler schreiben Code und kümmern sich um Programmfehler. Nach der Bereitstellung einer Anwendung in der Produktionsumgebung haben Entwickler generell Interesse daran, dass die Anwendung funktioniert und dass Feedback vom Kunden eingeholt wird, so dass 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. Es ist also Hauptaufgabe eines Entwicklers, ständig Änderungen in der Arbeitsumgebung vorzunehmen. Die Rolle eines traditionellen Entwicklers kann an der Fähigkeit gemessen werden, Änderungen herbeizuführen. Der Wert eines Entwicklers für die Organisation wird in aller Regel durch die Initiative und die Fähigkeit widergespiegelt, mit der neue Anwendungen und innovative Funktionen erstellt werden, anhand deren Benutzer produktiver sein können.

###### IT-Betrieb

Der IT-Betrieb, für den im Deutschen auch oft die englische Bezeichnung IT Operations verwendet wird, bzw. die IT-Administratoren sind dafür verantwortlich, dass alles optimal läuft. Dazu muss Hardware und Software zur Verfügung stehen und funktionieren. Zu den Hauptaufgaben eines IT-Administrators gehört es auch, den reibungslosen Ablauf von Datenverkehr, Infrastrukturpflege, Troubleshooting von Systemen und Datenbanken zu gewährleisten. Kurz gesagt besteht die Rolle eines klassischen IT-Betriebsspezialisten darin, eine stabile und optimale Infrastruktur bereitzustellen. Dazu müssen Änderungen minimiert werden, so dass Benutzern eine stabile und effiziente Infrastruktur zur Verfügung steht.

###### Widerspruch

Grundsätzlich haben Entwickler und IT-Betrieb ein gemeinsames Ziel, nämlich die Produktivität der Organisation zu gewährleisten. Dennoch ist offensichtlich, dass die Rollen dieser Abteilungen in Widerspruch zu einander 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 an der Umgebung um jeden Preis zu verhindern versucht. In Anbetracht dieser Tatsache zielen DevOps-Praktiken darauf ab, das Hin-und-Her zwischen diesen 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 kontinuierliche Lieferung von Software und Updates mit häufigem Release, oft sogar mehrmals am Tag, während vollständig neue Releases nur alle paar Monate oder ein Mal im Jahr herausgegeben werden. Lifecycle Management in einer DevOps-Kultur bietet auch den Rahmen für die Softwareentwicklung und erleichtert kontinuierliches Softwaremanagement.

Kontinuierliche Integration Eine Praktik bei der alle Arbeitskopien der Entwickler mehrmals am Tag in eine Hauptlinie kopiert werden.

Kontinuierliche Lieferung

Ein Ansatz, bei dem Software in kurzen Zyklen entwickelt wird, so dass sie jederzeit freigegeben werden kann.

ment with DevOps also provides the framework for software development and makes continuous management of the software even easier. Als Best Practices gelten die Erstellung eines knappen, fertigen Plans und der Anforderungen, durch die eine Idee zur Anwendung wird. Bei der Entwicklung von Software mit DevOps-Prinzipien sollte der gesamte Lebenszyklus der Anwendung bedacht werden. Dazu gehören die Softwarewartung, Softwareupdates, Außerbetriebnahme und Ersatz. Durch die Kombination dieser Faktoren wird beim DevOps-Lifecycle Management die Bereitstellung beschleunigt, die Workflow-Transparenz verbessert, hochwertigere Produkte geliefert und die Entwicklerzufriedenheit gefördert (Golden, 2014). Durch die konsequente Anwendung von Application Lifecycle Management anhand von kontinuierlicher Integration und kontinuierlicher Lieferung - Begriffe, die im folgenden Abschnitt behandelt werden - werden Unzulänglichkeiten in der Erstellungskette einer Lösung früh entdeckt. Automatisierte Qualitätssicherung und Verteilung der entsprechenden Produktartefakten trägt dazu bei, dass hohe Qualität und gute Nachhaltigkeit der Lösung gewährleistet ist. Im Begriff DevOps werden Entwicklung und Betrieb vereint, woraus Vorteile und Verbesserungen für eine Organisation entstehen.

### Dev und Ops im Betriebsalltag

Entwicklung und Betrieb sollten besser zusammenarbeiten. Das Zauberwort lautet DevOps, denn mit DevOps können agile Arbeitsmethoden in den Arbeitsalltag eines Unternehmens integriert werden. Heute werden Anwendungen immer mehr 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 Effektivität und Effizienz. Die dabei erzielte Transparenz und die Feedbackmöglichkeiten bringen jedoch zusätzliche Schwächen ans Licht. Das sollte nicht notwendigerweise auf Kosten der Qualität gehen. Der nächste Schritt ist es, Entwicklungs- und Betriebsteams im Rahmen der DevOps-Kultur einander näher zu bringen und deren Reaktion auf Änderungen zu beschleunigen. Es ist allerdings kein kleiner Schritt, der kulturelle Änderungen bei der Softwareentwicklung mit sich bringt, und ein Baustein auf dem Weg zur Cloud. Über die kontinuierliche Integration und Lieferung hinaus bekommt eine Organisation mit DevOps mehr Feedback und einen besseren Einblick in den gesamten Softwareentwicklungsprozess, 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 einfach Fragen, die nicht den gesamten Status abgreifen, sondern einen Hinweise darauf geben können, wo die Organisation steht:

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

DevOps

Wenn die Antwort auf diese Fragen Nein lautet, besteht noch keine DevOps-Kultur im Unternehmen. Wenn die Antwort auf eine oder zwei Fragen Ja lautet, hat die Änderung hin zur DevOps-Organisation begonnen.

###### DevOps einführen

Bei der Einführung des DevOps-Konzepts müssen ein paar Dinge beachtet werden. Wie aus der bisherigen Erläuterung bereits hervorging, geht es bei DevOps eher um die Kultur und die Vorgehensweisen als um die Organisation. Es ist in keiner Organisation einfach, die Kultur zu ändern. Genau wie bei der Einführung einer agileren Arbeitsweise, müssen die sogenannten soft Skills auch bei einer erfolgreichen Umstellung auf DevOps im Vordergrund stehen. Es ist unter Umständen relativ schwer, die Barrieren zwischen zwei historisch getrennten Organisationseinheiten, d.h. Entwicklung und Betrieb, zu überwinden. Eine erfolgreiche Integration wird jedoch mit hochwertigeren Apps und zufriedeneren Benutzern belohnt (Fazliu, 2018). Es gibt einige Werkzeuge und Techniken, die bei der Verknüpfung von Vorgehensweisen, so dass sie zu einem integralen Bestandteil des täglichen Arbeitsablaufs werden, helfen. Werkzeuge können auch zur Durchsetzung von Best Practices, z.B. zum Austausch von Problemlösungsdaten zwischen der Entwicklung und dem Betrieb, verwendet werden. 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 unersichtlichen Gründen nur schwer auf einen gemeinsamen Ansatz für die Definition von Anforderungen, die für Anwendungen und Projekte notwendig sind, einigen. Die Auswirkung solcher Uneinigkeiten auf den Fortschritt eines Projekts kann im Lauf der Zeit verheerend werden, und viele Stakeholder und Benutzer haben oft das Gefühl, dass sie nicht das System bekommen haben, was sei eigentlich gebraucht hätten. Natürlich ist es leicht, sich darüber zu beschweren, aber es ist wichtig, dass man versucht, 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 schon von Nutzen, dass DevOps selbst ein agiler Ansatz ist, der die Businessleute und die Entwickler einander näher bringt.

###### Kommunikation

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

Was die Kultur angeht, ist gegenseitiger Respekt notwendig. Auf Kommunikationsebene müssen die Wege verkürzt werden, denn die effektivste Kommunikation findet von Angesicht zu Angesicht statt, beispielweise wenn zwei Mitarbeiter gemeinsam an einem Whiteboard arbeiten. Natürlich muss der IT-Betrieb in der Lage sein, operative Anforderung formulieren zu können. Wenn diese bei den Entwicklern auf Verständnis treffen sollen, ist ein respektvoller Umgang miteinander äußerst hilfreich. Es ist keine große Überraschung, dass eine gemeinsame Sprache eine der Grundanforderungen ist.

###### DevOps-Pipeline

Eine der wichtigsten Voraussetzungen bei der Schaffung einer gemeinsamen Sprache mit dem Ziel bessere Kommunikation ist die Existenz eines Grundkonzepts, das von allen Stakeholdern, besonders Entwicklungs-, IT-Betriebs- und 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 Entwicklern und dem Betrieb, d.h. sie kann Entwickler und Betrieb enger zusammenbringen (Thedev, 2019). Die folgende Abbildung zeigt die DevOPs-Pipeline.



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

Planen

Bevor Entwickler mit dem Codieren beginnen, muss zunächst ein gründlicher Plan des gesamten Workflows stehen. Dabei spielen Projektmanager und Projektmanager eine wichtige Rolle. Es liegt in ihrer Verantwortung, einen Produktionsplan zur Führung des gesamten Teams durch den Prozess zu aufzustellen. Nach der Sammlung von Input und relevanten Informationen von den Benutzern und Stakeholdern wird der Auftrag in einzelne Aufgaben auf einer Liste geteilt. Durch dieses Aufbrechen in kleinere, überschaubare Teile können schneller Ergebnisse erzielt, auftretende Probleme abgearbeitet und auf unerwartete Änderungen reagiert werden. DevOps-Teams arbeiten an Sprints, die in der Regel kürzer, also ca. zwei Wochen, sind und während derer einzelne Teammitglieder die ihnen zugeteilten Aufgaben abarbeiten.

Entwickeln

Während dieser Phase beginnen die Entwickler mit der Codierung. Je nach Programmiersprache installieren Entwickler die notwendige integrierte Entwicklungsumgebung, Editoren und andere Softwaretools auf ihren Computern, um so effizient wie möglich arbeiten zu können. Manchmal müssen sich Entwickler Codierstils und -richtlinien aneignen, so dass ihr Code einem klaren Muster folgt, dass andere im Team leicht lesen und verstehen können.

DevOps

Wenn Entwickler 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 deshalb wichtig, da dabei Codierungsfehler zum Vorschein kommen, bevor sie durch die Pipeline geschickt werden und eventuelle größere Probleme verursachen. Typischerweise wird durch den Pull Request eine 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 Anwendung in Java im Gegensatz zu Anwendungen in Python kompiliert werden. Bei einem Codefehler wird der Build abgebrochen und der Entwickler wird davon benachrichtigt. Der ursprüngliche Pull Request schlägt dann ebenso fehl. Dieser Schritt findet jedes Mal statt, wenn ein Entwickler Code ins Repository schickt, so dass nur fehlerfreier Code in die Pipeline gelangt.

Testen

Bei Erfolg des Builds wird getestet. Das heißt, Entwickler führen manuelle und automatisierte Tests durch, um die Qualität des Codes weiter zu verifizieren. In vielen Fällen wird auch ein Akzeptanztest vom Kunden durchgeführt; dabei interagieren Menschen als Benutzer mit der Anwendung, so dass entschieden werden kann, ob weitere Änderungen am Code vorgenommen werden müssen, bevor er in die Produktion geht. Gleichzeitig werden oft automatische Tests durchgeführt: Sicherheitsprüfung der Anwendung, Suche nach Infrastrukturänderungen und auf Einhaltung der Best Practices, Performanz- oder Auslastungstests. Das Unternehmen muss entscheiden, was für die Anwendung wichtig ist und Tests zu diesem Zeitpunkt durchgeführt werden soll. In dieser Phase ist das Testen allerdings günstig, denn ein neuer Test kann, ohne den Entwicklerfluss zu unterbrechen oder die Produktionsumgebung zu stören, eingeführt werden.

Release

Die Release-Phase, im Deutschen auch Freigabephase, ist in DevOPs-Pipelines ein Meilenstein, 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 neue Funktionalität auszublenden, können Entwickler Funktionskennzeichnungen aktivieren, so dass Kunden diese erst dann sehen, wenn sie wirklich betriebsbereit sind. Als Alternative kann ein Unternehmen die Kontrolle über Builds behalten, die in die Fertigung geschickt werden. Es kann tägliche 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 Mitarbeiter einen Release in die Entwicklung genehmigen können. Die Technologie kann konfiguriert werden. Wie ein Unternehmen damit umgeht, bleibt ihm überlassen.

Bereitstellen

Das Programm kann in die Produktion gehen, wenn die Kompilierung den Bereitstellungspunkt 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, so dass die Funktion des neuen Codes geprüft werden kann. Bei wichtigen Updates wird häufig die Blau-Grün-Bereitstellung als Strategie verwendet.

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

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

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

Ausführen

Nun ist die letzte Version live und Kunden können sie benutzen. Das heißt das Betriebsteam sorgt dafür, dass alles reibungslos läuft. Je nach Hosting-Server und -Diensten wird die Umgebung automatisch an die Zahl der aktiven Benutzer angepasst. Das Unternehmen bietet den Kunden auch den Service sowie die passenden Tools dazu, Feedback zu sammeln, so dass die Weiterentwicklung des Produkts gewährleistet werden kann. Kunden wissen am besten, was sie brauchen, und Benutzer sind die besten Tester, die weit mehr Zeit mit der Anwendung verbringen, als in einer DevOPs-Pipeline je möglich.

Überwachen

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

###### DevOps-Best Practices

Eine Organisation braucht, um die Vorteile von DevOps voll zu nutzen, saubere Implementierungspläne. Es wurde bereits ausgeführt, was DevOps ist, warum es notwendig ist und welche Schritte die DevOps-Pipeline umfasst. Jetzt können wir uns auf einige Best Practices von DevOps konzentrieren (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 von Konfigurationsmanagement liegt beispielsweise darin, dass neue Umgebungen leichter eingerichtet werden können, dass die Konfiguration der Produktionsumgeben vereinheitlicht wird und dass Zeit und Ressourcen bei der Softwareerstellung gespart werden können, da die nachstehend erläuterte Praxis des Infrastructure as Code eingesetzt werden kann, um neue Dienste von Grund auf zu erstellen.

DevOps

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 und IT-Betriebsmitglieder, wodurch weniger und kürzere Feedbackzyklen und schnellere Updates möglich sind. DevOps-Teams tragen gemeinsam die Verantwortung für die von ihnen angebotenen Dienste, für den Code und die Rufbereitschaft. Vorfälle werden sowohl während der Freigabe als auch danach schneller entdeckt und bearbeitet, denn Softwareentwickler und IT-Fachleute stehen während des gesamten Auslieferungszyklus zur Verfügung und leisten auch danach noch auf Abruf Bereitschaftsdienst.

Kontinuierliche Integration

Bei der heutigen Anwendungsentwicklung arbeiten mehrere Entwickler 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 Entwicklern erstellt wurden, können zu Unvereinbarkeiten führen, wenn sie gleichzeitig zusammengeführt werden. Dieses Problem kann dadurch verstärkt werden, wenn Entwickler ihre eigenen lokal eingerichteten Entwicklungsumgebungen haben, statt gemeinsam in cloudbasierten integrierten Entwicklungsumgebungen zu arbeiten. Anhand der kontinuierlichen Integration können Codeänderungen öfter, ja bis zu täglich in einen gemeinsamen Entwicklungszweig bzw. einen Hauptentwicklungszweig zusammengeführt werden. Wurden alle Änderungen zusammengeführt, werden sie durch automatisierte Anwendungsbuilds 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).

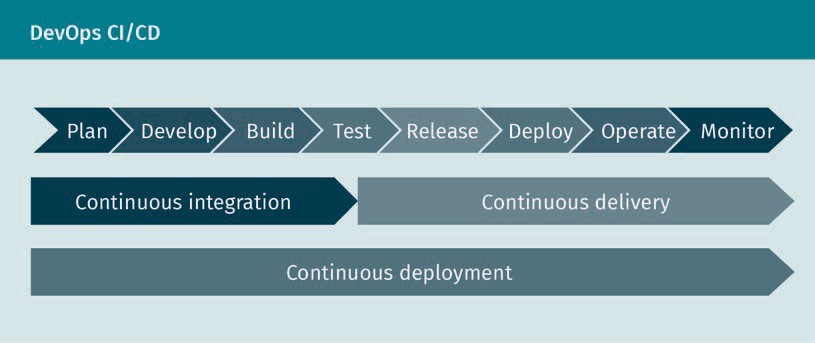
Kontinuierliche Lieferung

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 effiziente, kontinuierliche Lieferung zu gewährleisten, muss bereits 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

Man kann die kontinuierliche Software-Verteilung als Erweiterung der kontinuierlichen Lieferung betrachten, 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 kontinuierliche Software-Verteilung, dass Änderungen eines Entwicklers an der App innerhalb von Minuten nach der Erstellung live gehen können. Sie müssen dazu nur die automatisierten Tests bestehen (Patel, 2020). Dadurch wird die kontinuierliche Integration von Benutzerfeedback viel einfacher. All zusammenhängenden Praktiken der kontinuierlicher 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.



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 werden 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 uralte Problem, dass 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. Dadurch wird menschlichen Fehler vorgebeugt, wodurch wiederum die Softwareentwicklung und der Infrastrukturbetrieb beschleunigt und vereinfacht werden.

Testautomatisierung

Durch automatisiertes Testen jeder Codebasis können Entwickler mehr Tests durchführen, steigt die Testgeschwindigkeit und 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 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.).

DevOps

###### 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 Entwicklung und die andere Hälfte mit 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).

### Auswirkungen auf die Team- und Entwicklungsstruktur

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 schicke 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 natürlich in jedem Unternehmen eine andere Form an. Das machen wir uns bei der Analyse der verschiedenen Modelle zu Nutzen. Was am besten für ein 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 hat die Teamführung und welche Branchenerfahrung bringen diese leitenden Personen mit? Haben Entwicklung und Betrieb dieselben Prioritäten oder werden sie von der Erfahrung des jeweiligen Leiters 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 in der DevOps-Welt viel Aufsehen mit der Idee der Zwei-Pizza-Teams erregt: Das leistungsstärkste Team ist klein genug, um von zwei Pizzen satt zu werden(Buchanan, 2019). Kleinere Teams führen zu lose verbundener Architektur und Microservices, die von leistungsfähigen

Silos

Bei diesem Managementsystem kommt es zu einer Struktur, bei der Teams auf die eigenen Ziele konzentriert sind, anstatt auf Unternehmensziele ausgerichtet zu sein.

DevOps-Organisationen bevorzugt werden. Eine Reorganisation in kleinere Teams ist jedoch leichter gesagt als getan und kann eine Organisation überwältigen. Ein Ansatz, mit dem der Prozess begonnen werden könnte, ist die Bildung einer Portfolioorganisation. Portfolioteams bestehen aus Entwicklern, IT-Fachleuten, Testern, klassischen Projektmanagern usw. Sie können funktionsbezogen oder rollenbezogen arbeiten, sollten jedoch alle das Ziel haben, die Fähigkeit der Firma zu verbessern, schneller Wert an die Kunden liefern zu können. Eine Firma könnte beispielsweise einen Vizepräsidenten oder einen Technologiedirektor haben, der eine Portfoliostruktur für die Organisation entwirft. Nachdem der Entwurf steht, managt der Vizepräsident einige kleine, von Teamleitern geführte DevOps-Teams, die an den Firmenzielen ausgerichtet sind.

###### Führung

Angenommen in einem Unternehmen existiert ein hoher Grad an Instabilität und es besteht 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 Veränderungen der Kultur hin zu DevOps auftreten, anzugehen. Veränderungen in einer Organisation lassen sich oft schwer implementieren: Der Ansatz muss unternehmensweit erfolgen und mehrere Abteilungen müssen sich auf einen Implementierungsansatz einigen. Eine Organisationsänderung ist nie einfach, sie ist jedoch besonders für die schwer, die von vornherein nicht gut miteinander arbeiten. Völlig fehlende oder schwache Änderungsbereitschaft sowie niedrige Einsatzbereitschaft der Mitarbeiter gelten als Misserfolgsindikatoren.

Transformationale Führung ist ein Führungsstil, bei dem Mitarbeiter gefördert, bemächtigt und motiviert werden, Änderungen für Organisationswachstum durchzuführen und den möglichen Erfolg des Unternehmens mitzugestalten (Wikipedia, 2020). Ein derartiger Führungsstil kann je nach Reaktion der Teammitglieder auf die Änderungen hin zur DevOps-Kultur erfolgversprechend sein.

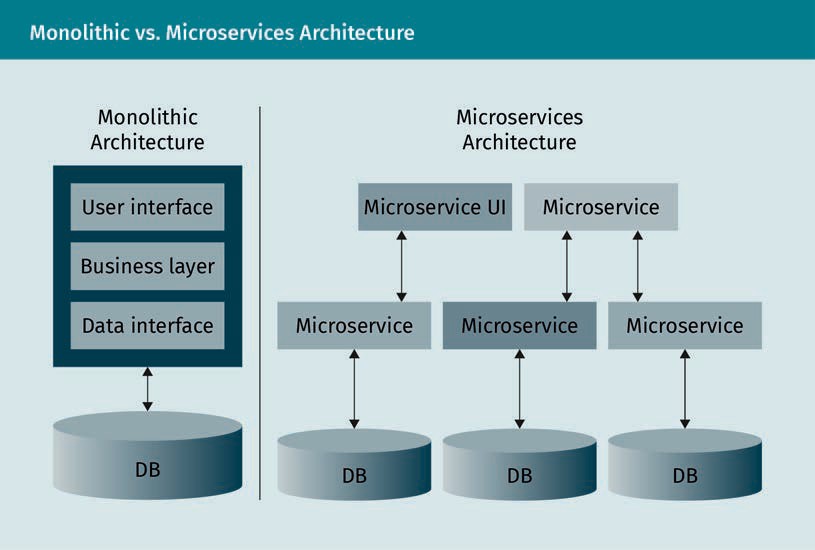
###### Die richtige Mischung

Auf dem Weg zu DevOps-Teams sollten als erstes die Qualifikationsdefizite analysiert werden. Dabei werden zunächst die Rollen und Fähigkeiten, die die Mitarbeiter haben müssten, um die Teamziels umsetzen zu können, bestimmt. In diesem Sinne muss eruiert werden, welche Teammitglieder derzeit welche Stellen innehaben, wie viele zusätzliche Mitarbeiter neu angestellt werden müssen und welche Fähigkeiten neue Mitarbeiter mitbringen müssen, um das Defizit zu beheben. Dabei spielen nicht nur die fachlichen Qualifikationen oder Aufgaben eine Rolle. Ein Manager muss auch die zusätzlich notwendigen interpersonellen Fähigkeiten und persönlichen Eigenschaften ermitteln, die zu einer guten Mischung im Team führen.

DevOps

###### Microservice-Architektur

Apps können auf verschiedene Arten und Weisen entwickelt werden. Anstatt beispielsweise ein Projekt als Ganzes, also monolithisch, zu implementieren, könnte eine Verteilung der Aufgaben in kleinere Pakete (Microservices) sinnvoll sein. Dieses Konzept passt nicht nur auf die 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 Entwickler sich 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 Trends, die heute Firmenkulturen mitbestimmen. Bei beiden geht es um Aktivitäten, durch die eine Organisation belastungs- und leistungsfähiger wird. Man kann sagen, dass die Grundlage für hervorragende Microservices eine ausgezeichnete DevOps-Kultur ist. Der Begriff der Microservice 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 geteilt 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).



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 Integration der Entwicklungs- und Betriebsteams, das heißt die 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, bietet Anwendungen, die die Infrastruktur optimal ausnutzen können, und führt zu Prozessen, anhand deren diese Anwendungen schnell und mit hoher Qualität entworfen und implementiert werden können.

###### Twelve-Factor-App

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

Sie befolgt 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.

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 aufs 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 aufs 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 weniger abhängig die CI/CD-Pipeline von externen Anwendungsserver 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.

DevOps

1. 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.
2. 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.

### 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, legen wir hier mindestens den Grundstein. Wie zuvor behandelt, 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 weitverbreitendem 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 den Setup mit Git, Jenkins, Docker und Kubernetes.

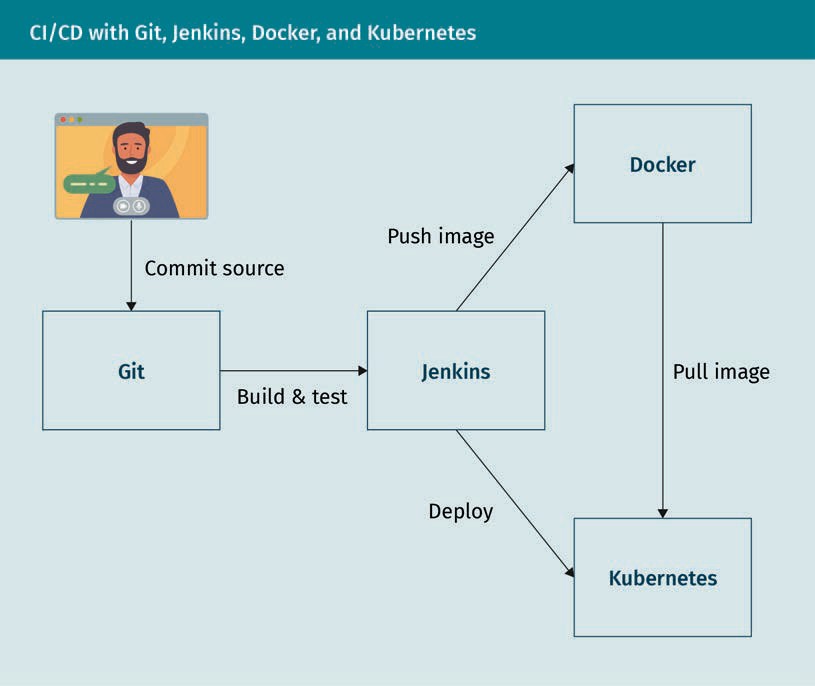
Prozessmodell Ein Modell aus einer Sequenz von Phasen und Aufgaben, die den gesamten Systemlebenszyklus abdecken.

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

Container

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

VersionsverwaltungssystemEin System, das Änderungen an einer Datei oder einem Dateiensatz speichert, so dass bestimmte Versionen später wieder abgerufen werden können.



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.

DevOps

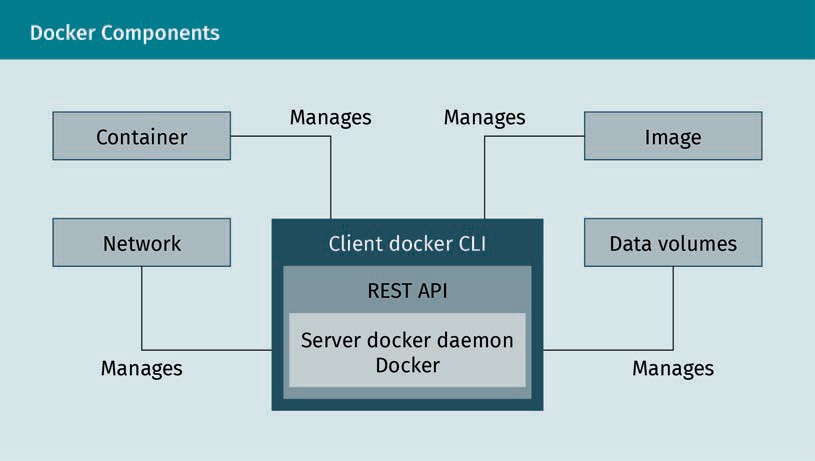
###### Docker

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 Anwendungsausführung 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-spezifischen Ausdrücke sollte man kennen (Docker, o. D.-b).

* + Dockerfile: Eine Textdatei mit Befehlen zur Erstellung eines Image.
  + 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.

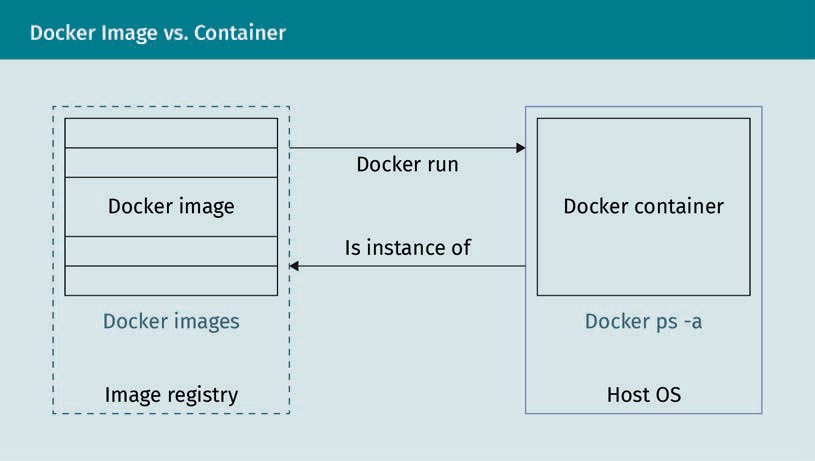
Das folgende Diagramm zeigt die Hauptkomponenten des Docker-Systems.



Ein Container wird durch ein Image gestartet (Container bestehen aus Images). Ein Image ist ein auszuführendes Paket, das alles, was zur Ausführung nötig ist (d.h. den Code, ein Runtime-System, Bibliotheken, Umgebungsvariablen und Konfigurationsdateien), enthält.

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

DevOps



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

FROM python:3

ADD my-script.py /

CMD [ "python", "./my-script.py" ]

In der obigen Dockerfile-Datei weist FROM 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 .

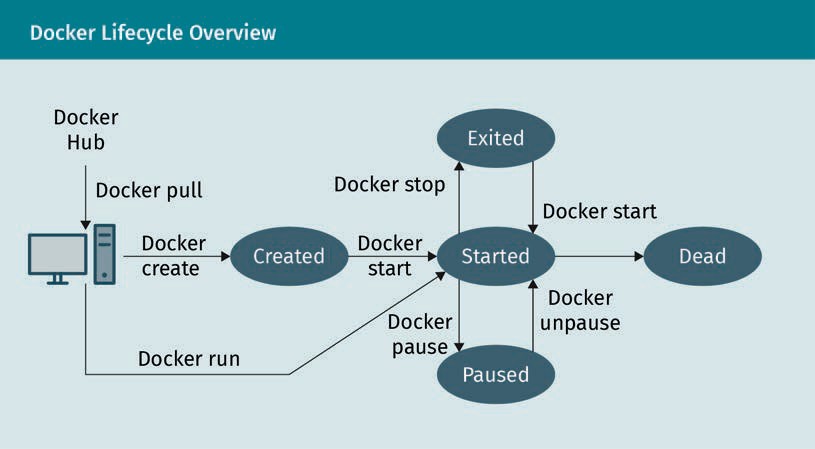
Nach der Image-Erstellung kann ein Container mit folgendem Befehl ausgeführt werden:

$ 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, das dazu verwendet wir, 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.



Docker Hub ist eine Speicherplatz für Docker-Images für öffentlichen Zugriff und für Entwickler, um schnell neue und zusammengesetzte Anwendungen zu erstellen. Mithilfe der Push- und Pull-Befehle können Images von der Docker Hub ausgelesen sowie auf sie zurückgeschrieben werden (Docker, o.D.-c).

###### Ansible

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

DevOps



Ansible ist Software für die zentrale Verwaltung von verteilten Servern. Die Community-Version von Ansible ist lizenzfreie 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 lizenzpflichtig sind und beispielsweise Dashboards oder Workflows anbieten. Ansible ist mit Puppet und Chef eines der bekanntesten Softwareprodukte zur Verwaltung von verteilten Servern. Im Vergleich zu Puppet und Chef hat Ansible jedoch mehrere Vorteile (Arora, 2020):

* + Für Ansible ist keine zentrale Komponente nötig. Ein Computer reicht zum Zugriff auf Server aus, die über SSH (bzw. dem 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. aus 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, auf dem Linux installiert ist, verwendet werden. Das könnte sowohl die Workstation des Linux-Administrator als auch ein anderer Computer sein, von dem aus die Server, die verwaltet werden müssen, erreicht werden können.
    - Netzwerk: Die Server, die von Ansible auf der Verwaltungsinstallation verwaltet weden 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 Konfigurationsverwaltung erleichtern und ein Framework für die Bereitstellung auf mehrere Computer, das mehrmals wiederholbar ist, ermöglichen. Das folgende Playbook beschreibt beispielsweise die Installation von NGINX:

- 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 die restliche Infrastruktur, z.B. Load Balancer, Datenbanken, gedacht (Terraform, o.D.). Diese Spezialisierung heisst, dass Terraform für bestimmte Aufgaben ganz 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 Cloudumgebung dynamisch geändert werden. Dadurch hat der Kunden die Möglichkeit, je nach Bedarf mehr oder weniger Ressourcen zu nutzen. Anbieter bieten diese Flexibilität anhand von APIs, durch die geleaste 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.

DevOps

### Skalierbare Umgebungen

Skalierbarkeit ist ein Kernziel der modernen Infrastruktur, da eine Organisation, die konsequent skalieren kann, 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 Kommunkationsmöglichkeiten, Ausrichtung auf Hochleistung, mehr Möglichkeite 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). 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 ausgegelichenen Auslastung in der Infrastruktur
  + Lastenausgleich und -routing
  + Überwachung des Containerstatus
  + Konfigurierung von Anwendungen nach den Containern, in denen sie ausgeführt werden sollen
  + Absicherung der Containerinteraktion

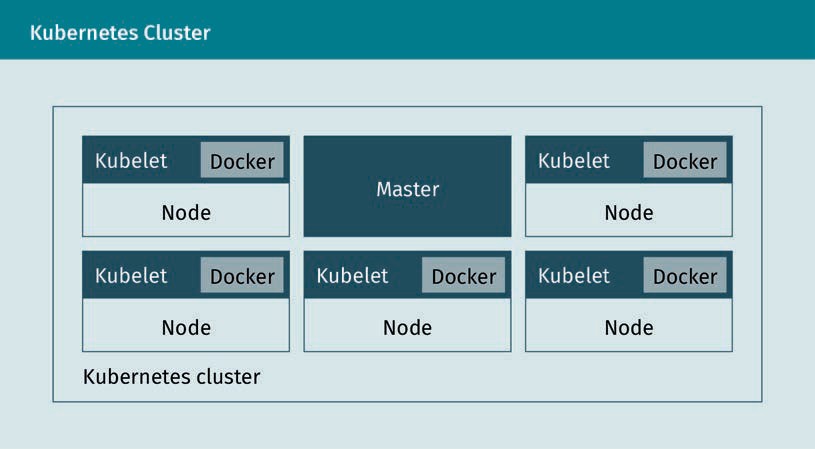
Tools zur Container-Orchestrierung bieten ein Framework für die groß angelegte Verwaltung von Containern ud Microservicearchitekturen. Es gibt viele Container-Orchestrierungs-Tools, die zum Container-Lifecycle Management verwendet werden können. Zu den am häufig eingesetzten gehören Kubernetes, Docker 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 Anwendungen 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 notewendig ist. Workloads sind mit Kubernetes auch portierbar und Lastenausgleich ist möglich, da Anwendungen ohne Neuentwicklung übertragen werden können. Das folgende Diagramm zeigt einen Kubernetes-Cluster:



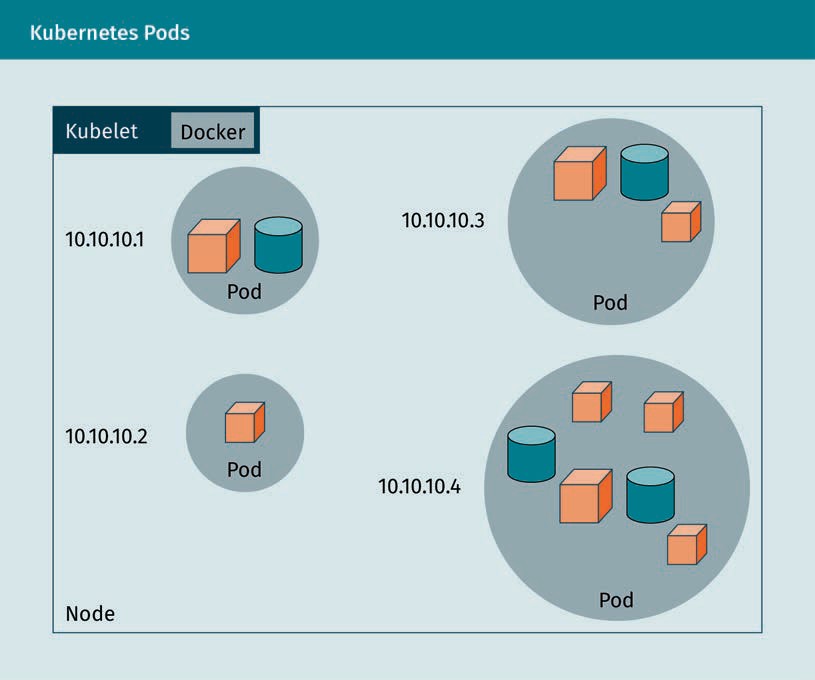
Ein Kubernetes-Cluster besteht aus zwei Ressourcenarten:

* 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 der 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 Agent 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 Anwendunge 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 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 mehrerer Container mit gemeinsamen Speicher- und Netzwerkressourcen; Pod ist auch eine Spezifikation für die Ausführung von Containern (Kubernetes, 2020). 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.

DevOps

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.



###### 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 Software, 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 zugänglich (Docker, o. D.-d). Docker Swarm beruht auf einer Master-Slave-Architektur. Jeder Docker-Cluster, der sogenannte Swarm, besteht aus mindestens einem Managerknoten und beliebig vielen Worker-Knoten. Während der Swarm-Manager für die Verwaltung 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 Definition von Aufgaben, die im Cluster ausgeführt werden müssen, verwendet wird. Jeder Dienst besteht aus einer Reihe von einzelnen Aufgaben, die alleine in einem separaten Container auf den Knoten im Cluster abgearbeitet werden. Bei der Erstellung eines Dienstes wird 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 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 hinzugefügt, teilt der Swarm-Manager diesem unverzüglich eine Augabe für den globalen Dienst zu. Globale Services eignen sich beispielsweise für Monitoring- oder Protokollierungsanwendungen. 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 intelligent 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, die 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 Lerneinheit wurde 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 DevOps-Pipeline, Konfigurationsmanagement, Releaseverwaltung, kontinuierliche Integration, kontinuierliche Lieferung, Infrastructure as Code, Testautomatisierung und kontinuierliches Monitoring, wurden erläutert. Die Auswirkungen von DevOps auf die Team- und Entwicklungsstrukturen wurden anhand von Teamgröße, Teamführung, Microservice-Architektur und die Prinzipien der Twelve-Factor-App dargelegt.

DevOps

Diese Einheit war auch 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.



# Lerneinheit 3

## Softwareentwicklung

#### LERNZIELE

Nach Abschluss dieser Lerneinheit wissen Sie...

... über verschiedene Testverfahren im Lebenszyklus der Softwareentwicklung Bescheid.

... über verschiedene Testansätze während der Entwicklung, wie testgetriebene und verhaltensgetriebene Softwareentwicklung Bescheid.

... was kontinuierliche Integration (CI), kontinuierliche Lieferung (CD), kontinuierliches Testen (Cte) und kontinuierliches Training (CT) ist.

... wie ein CI/CD-Ansatz zur Verbesserung von Machine-Learning-Systemen verwendet wird.

... wie Änderungen in einem Softwareentwicklungsprojekt mithilfe von Tools nachverfolgt werden.

... was integrierte Entwicklungstools (IDEs) sind und kennen einige Beispiele.

DL-E-DLMDSSEDIS01-U03

1. Softwareentwicklung

### Einführung

Nach Abschluss der Entwicklung einer Machine Learning-Lösung (ML) wird die Software nicht sofort den Kunden in der Produktionsumgebung zur Verfügung gestellt. Davor kommt noch ein Schritt, nämlich das Testen. Testn ist eine funktionsübergreifende und kontinuierliche Aktivität des ganzen Teams. Im Idealfall arbeitet das Testteam mit den Entwicklern und Benutzern 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 relavanten 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 funktionierender Software, d.h. weniger Programmfehler und geringere Supportkosten, und legt ein Framework für Entwicklungspraktiken fest. Am Anfang dieser Lerneinheit werden verschiedene Testarten aus der Sicht der verschiedenen Stakeholder eines Projekts besprochen. Im Anschluss befassen wir uns damit, wie Testen und Entwickeln gleichzeitig stattfindet. Ein Szenario für die Entwicklung einer Machine Learning-Lösung, also einem nicht-deterministischen System, ist sehr komplex und davor müssen wir die konventionellen Testverfahren wiederholen. Auch die Automatisierung der Entwicklungspipeline wird behandelt. Und schließlich besprechen wir die automatisierte Entwicklung, bei der es sogar noch wichter ist, Änderungen nachzuhalten.

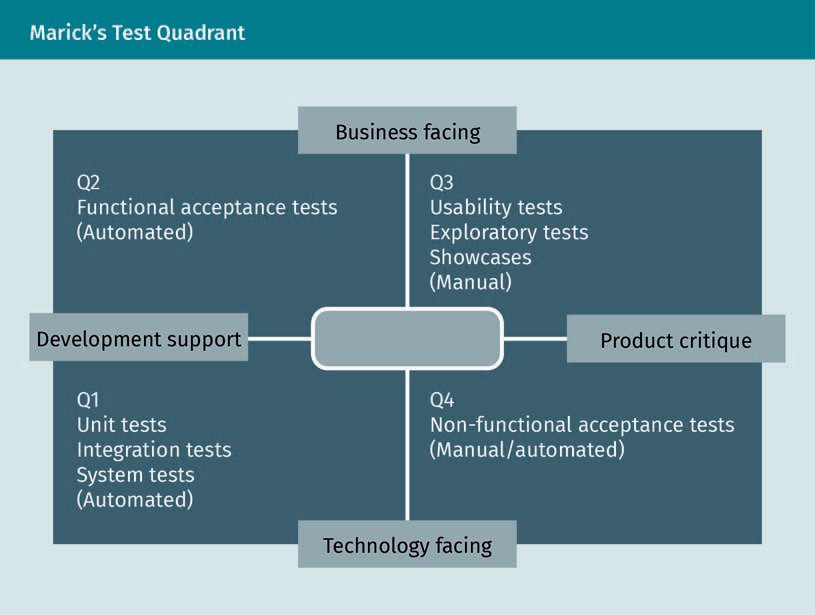
### Testparadigmen und Überwachung

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). Tests fallen in zwei Dimensionen:

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

Diese vier Szenarien werden nun im Detail besprochen.

Softwareentwicklung



Quadrant 1 (Q1)

Unter den technologieorientierten Tests für das Entwicklungsteam sind 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 is notwendig. Durch die Isolation kann das Modultesten schnell abgewickelt werden. Allerdings kann durch den Modultest nicht die tatsächliche Funktionalität im Zusammenspiel mit anderen Softwarekomponenten gestestet werden. Daher muss eine andere Testreihe, nämlich die Integrationstests, angesetzt werden. Bei einem Integrationstest wird eine Bandbreite an Funktionalität in verschiedenen Modulen des Softwareprodukts getestet. Dabei wird 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 getestet, um die Kompatibilität mit den Kundenanforderungen, die in der Software-Anforderung (abgekürzt SRS für Software Requirements Speciﬁcation) festgelegt ist, beurteilt. Beim Integrationstest wird ein funktionaler Test auf einer Reihe von Modulen und ihre Interaktionen ausgeführt. Beim Systemtesten wird das System insgesamt getestet. Systemtesten kommt nach dem Integrationstesten. Vor der Bereitstellung einer Software-Lösung in der Produktionsumgebung ist 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 In der Qualitätssicherung von Software eine Sammlung von Software und Testdaten, die zum automatisierten Testen eines Programms unter verschiedenen Umgebungsbedingungen und zur Überwachung des Verhaltens und der Outputs verwendet wird.

Funktionaler Test 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.

Testingenieur Eine Person in einem Softwareprojekt, die während des Softwareentwicklungszyklus verantwortlich für Definition, Ausführung und Berichterstellung der Testverfahren ist.

Quadrant 2 (Q2)

Tests, die das Entwicklungsteam in geschäftsorientierten Aspekten unterstützt, werden als funktionale Akzeptanztests bezeichnet. Ein Akzeptanztest gewährleistet, dass vorgegebene Kriterien, wie Funktionalität, Kapazität, Usability und Sicherheit, erfüllt werden. Ein Akzeptanztest für eine Funktion wird als funktionale Akzeptanztest bezeichnet. Wenn eine Software einen derartigen Test besteht, können das Entwicklungsteam und die Benutzer davon ausgehen, dass die Funktionalität des entwickelten Moduls richtig implementiert wurde. Auch das End-to-End-Testen (E2E-Testen), wenn Entwickler testen, ob die gesamte Software von Anfang bis Ende wie erwartet funktioniert fällt in diese Kategorie (Q2). Bei der Testkonzipierung werden die Systemabhängigkeiten definiert. Der Test gewährleistet, dass alle Teile der Softwarelösung nach Spezifikation zusammenwirken. Das Hauptziel des e2E-Testen ist es, durch die Simulation tatsächlicher Benutzerszenarien die Software aus der Sicht der Benutzer zu testen (Katalon, 2020). Beim Systemtesten wird nur das Softwaresystem selbst getestet. Beim E2E-Testen wird das System in Verbindung mit den externen Systemen getestet.

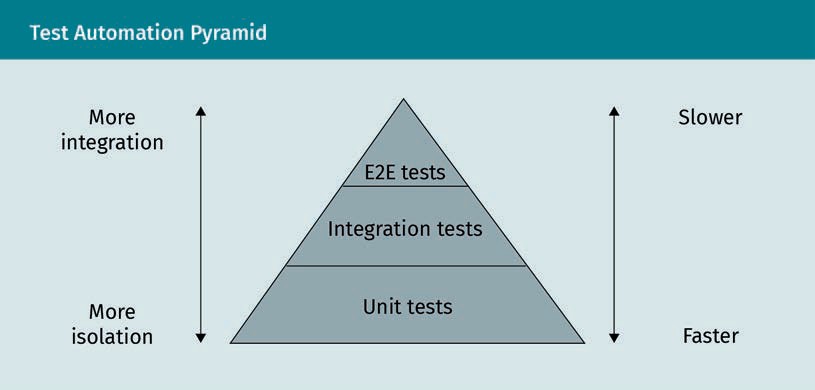
Quadrant 3 (Q3)

Geschäftsorientierte Tests zur Beurteilung des Projekts sind meistens manuelle Tests, durch die geprüft wird, ob das Produkt den vom Kunden erwartenden Wert liefern kann. Dabei werden nicht nur die Spezifikationen, sondern auch die Definition der Spezifikationen geprüft. In diese Testkategorie fallen Showcases, exploratives Tests und Usability-Tests. Am Ende einer Iteration stellt das Entwicklungsteam den Benutzern die neue Funktionalität im Rahmen eines Showcase vor. Diese Funktionalitätsdemo während der Entwicklungsphase beugt Missverständnisse oder Spezifikationsprobleme vor. Ein weiteres Testverfahren, das explorative Testen, wurde von J. Bach (2003) als manuelles Testen definiert, bei dem „der Testingenieur aktiv das Testdesign während der Testausführung kontrolliert und die gewonnen Informationen zur Entwicklung neuer, verbesserter Tests verwendet“ (S. 2). Exploratives Testen führt zu einer Reihe neuer automatisierter Testverfahren. Und schließlich werden Usability-Tests ausgeführt, um nachzuvollziehen, wie Benutzer anhand der entwickelten Software leicht die definierte Ziele erreichen können.

Quadrant 4 (Q4)

Technologieorientierte Tests zur Beurteilung des Projekts verifizieren die nicht-funktionellen Kriterien des Softwaresystems, z.B. Kapazität, Verfügbarkeit und Sicherheit. Dazu wurde das nicht-funktionelle Akzeptanztesten entwickelt. Diese Testart könnt auch voll automatisiert werden, sie wird allerdings wendiger häufig als das funktionale Akzeptanztesten und zum Schluss 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 nachfolgende 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 werden.

Softwareentwicklung



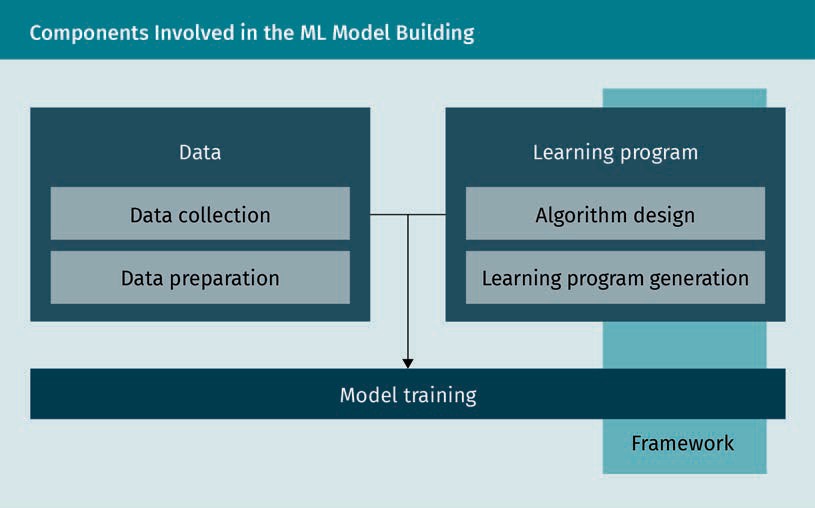
###### Testen von Machine Learning-Modellen

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 von dem ab, was das System in den vorherigen Transaktionen gelernt hat. Daher sollte der Tester 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 verwendeter Mechanismus, um den Erfolg des Tests zu messen (Kaner, 2004). Dabei wird das Output des zu testenden Systems, das als Input für ein bestimmtest Testszenario vorbereitet wurde, mit dem Ergebnis, das das Produkt (nach Einschätzung des Orakels) liefern sollte, verglichen.

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 all genannten Komponenten und die entsprechenden Interaktionen testen.

Nichtdeterminismus Begriff aus der Computerwisschenschaft, bei dem Algorithmen oder Maschinen nicht nur genau eine Berechnung auf ein bestimmtest Input hin (deterministisch) durchführen können. Es gibt dagegen mehrere Möglichkeiten für den Übergang zum nächsten Zustand für dasgleiche Input.

TensorFlow Eine End-to-End- und Open-Source-Plattform für Machine Learning mit einem umfassenden, flexiblen ökosystem aus Tools, Bibliotheken und Community-Ressourcen. Damit können Forscher den Stand der ML-Technik weiter vorantreiben, und Entwickler können leicht ML-Anwendungen bauen und bereitstellen (TensorFlow, o.D.).



Testen von Daten

Beim Testen von Datenkomponenten eines ML-Modells sollten Testingenieur z.B. 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 Trainings- und Testdaten eine Verzerrung (auch auf Deutsch Bias genannt) besteht (Zhang et al., 2020). Obwohl es Aufgabe des ML-Entwicklers st, die richtigen Testfälle zu konzipieren, gibt es allgemeine Richtlinien (Heck, 2020).

* Datentyp und -schema: ML-Ingenieure überprüfen Format, Art und Schema der Daten stets auf Datenqualität und -integrität hin. Anhand dieses Tests wird sicher gestellt, dass die Daten keine schlecht formatierten Inputs enthalten (Kim et al., 2018).
* Implizite Constraints: ML-Ingenieure 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 Datenvalidierungssystem 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 Metrik Richtigkeit gehören Accuracy (Genauigkeit), Präzision (Precision) und Recall (Abruf). Der Unterschied zwischen Precision and Recall kann am besten am Beispiel eines ML-Algorithmus erklärt werden.

Softwareentwicklung

Der Algorithmus teilt 100 Tumor 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, 2020b). 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 faires ML-Modell liefert faire Vorhersagen über verschiedene demographische Gruppen hinweg (Dwork, 2012). IBM hat ein Tool für ML-Fairness entwickelt, das AI Fairness 360 (IBM, o.D.-a). Es wird von ML-Entwicklern verwendet, um Diskriminierung und Zerrung in ML-Modells im gesamten AI-Anwendungslebenszyklus zu prüfen, zu protokollieren und auszugleichen. Ein ML-Modell sollte aus interprätierbares Modell verstanden werden, wenn ein Beobachter bzw. Benutzer den Grund der vom Modell getroffenen Entscheidungen verstehen kann. Es gibt ein paar Tools zur Interprätierbarkeitsprüfung, z.B. das Modul Machine Learning Interpretability von H2O (H2O, o.D.).

###### ML-Modell-Überwachung

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

1. Funktionsüberwachung zur Sicherstellung der langfristigen Stabilität eines Modells und der Datenvarienten 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 unbekanntem Bias

Amazon SageMaker Model Monitor ist ein Beispiel für ein ML-Monitoring-Tool, anhand dessen Entwickler das Phänomen des Konzeptdrift, 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.

### 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 geteilt werden. Jede Phase umfasst Aktivitäten mit dem Ziel, die Planung und Verwaltung des Projekts effizienter zu gestalten. Das Wasserfallmodell ist eine der klassischen Methode, während agile Modelle, wie Kanban und Scrum, die etwas moderneren Ansätze sind (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)

Konzeptdrift

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

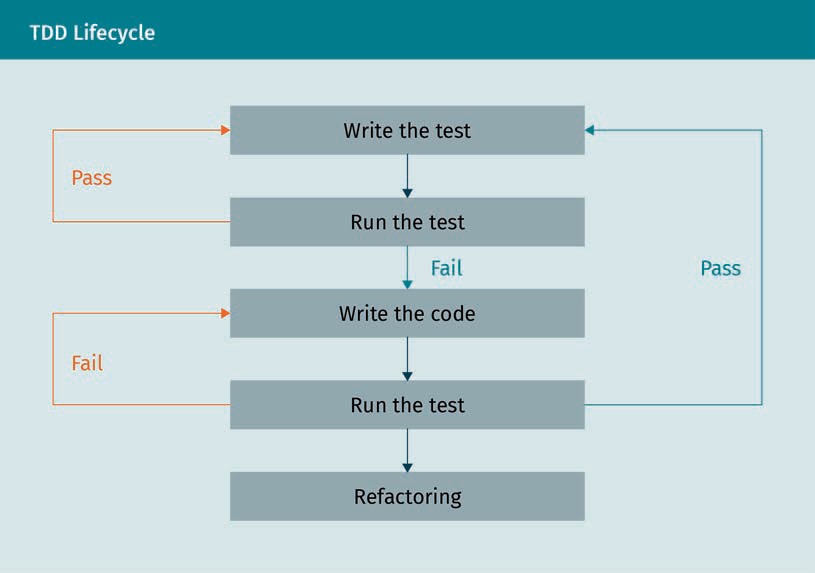
Agiles ModellEin Modell, bei dem sich Spezifikationen, Anforderungen und Code ständig durch die Zusammenarbeit funktionsübergreifender Teams ändern.

RefactoringDie Bereinigung und Vereinfachung bestehenden Codes im Softwareentwicklungszyklus, ohne dass dabei das Verhalten und die Funktionalität verändert wird.

###### Testgetriebene Entwicklung (TDD)

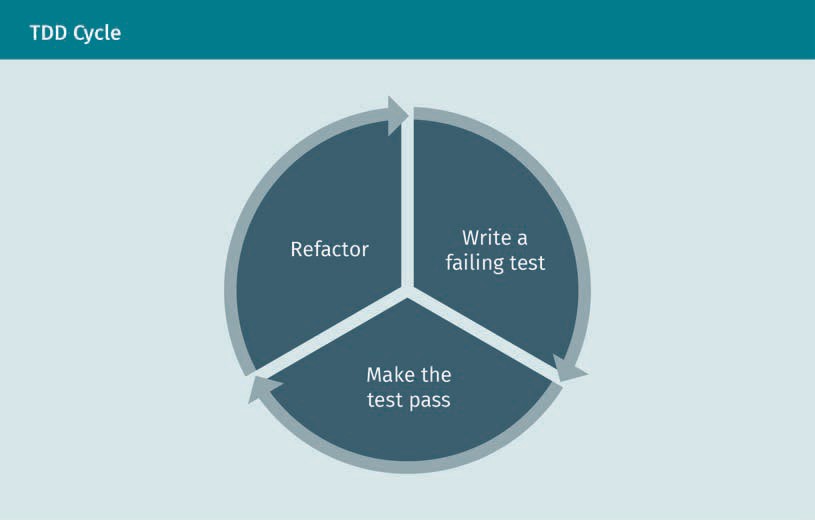
Bei der testgetriebenen Entwicklung werden Tests aus der Sicht des Entwicklers ausgeführt. Dabei werden von einem Qualitätssicherungsingenieur zunächst Testfälle für alle Funktionen der Software entworfen und geschrieben. Dabei sol eine einfache Frage beantwortet werden, nämlich ist der Code gültig. Der Hauptzweck dieser Technik ist es, den Code nur dann zu ändern oder nur dann neuen Code zu schreiben, wenn der Test fehlschlägt. Dadurch wird die Duplizierung 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 kann in folgende Schritte unterteilt werden (Beck, 2014):

1. Test schreiben. Der Test sollte bestimmte Funktionen definieren (oder verbessern). Daher sollte der Testentwickler die Produktanforderungen kennen. Das ist einer der großen Unterschiede zwischen einem klassischen Entwicklungsansatz und testgetriebener Entwicklung: erst wird der Test und dann der Code geschrieben.
2. Test ausführen und sicherstellen, dass er auch fehlschlägt, denn es gibt ja noch keine Funktionalität.
3. Code schreiben und zwar für die Funktionalität, die den bereits geschriebenen Test bestehen soll. Der Code sollte so einfach geschriebern sein, dass der Test auf jeden Fall funktioniert.
4. Test am neu geschrieben Code ausführen. Das Ziel sollte ein „Pass“ sein.
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:

Softwareentwicklung

Zusammenfassend läßt sich sagen, dass bei der testgetriebenen Entwicklung das Muster lautet: Test mit dem Ziel Fail schreiben, Test so ändern, dass am Ende ein Pass steht, Refaktoring und Wiederholung dieser Schleife (Freeman & Pryce, 2012).



###### Behavior Driven Development (BDD)

Beim Einsatz von testgetriebener Entwicklung in Projekten in verschiedenen Umgebungen kann es zu Verwirrung und Missverständnissen kommen. Programmierer 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 mit Fail abgeschlossen werden soll (North, 2020). Ein Ansatz, um dem gegenzusteuern, ist die verhaltensgetriebene Entwicklung (BDD), eine agile Softwareentwicklungspraktik nach Dan North (2006). Dabei wird angestrebt, dass alle 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 Benutzer hat gültige Anmeldeinformationen eingegeben.
  + Wenn: Ein Benutzer klickt auf die Anmeldeschaltfläche.
  + Dann: Die erfolgreiche Validierung wird durch eine Benachrichtigung angezeigt.

Durch die Verwendung von Klartext (hier in Deutsch) können alle Stakeholder (Entwickler und Kunden) das Funktionsverhalten nachvollziehen. Gherkin ist eine fachspezifische Sprache für Beispiel in der „Gegeben/Wenn/Dann“-Struktur und in Klartext-Dateien, die als Feature-Dateien bezeichnet werden. Feature-Dateien beschreiben, wie System und Benutzer miteinander interagieren. Ein einfaches Beispiel einer Feature-Datei in Gherkin könnte folgendermaßen aussehen:

Feature: Rechner

Szenario: “+” sollte zur laufenden Summe addiert werden Gegeben die laufende Summe ist “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 Kunden 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 mit verschiedenen Perspektiven an der Konzipierung von Akzeptanztests teil, bevor die relevante Funktionalität implementiert wird. Diese Akzeptanztests spiegeln die Perspektive des Benutzers wider. In der folgenden Tabelle werden die wichtigsten Unterschiede der drei Methoden, TDD, BDD und ATDD aufgeführt (Unadkat, 2020).

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

Softwareentwicklung

|  |  |  |  |
| --- | --- | --- | --- |
|  | TDD | BDD | ATDD |
| Hauptfokus | Modultests | Anforderungen aufgrund des Systemverhaltens verstehen | Schreiben von funktionalen Akzeptanztests |
| Teilnehmer | Entwickler | Entwickler, Kunden, Qualitätssicherungsingenieure | Entwickler, Kunden, Qualitätssicherungsingenieure |
| Sprache | Ähnlich wie der eigentliche Code | Natürliche Sprache | Natürliche Sprache |

Am Ende dieses Abschnitts gehen wir noch kurz auf die Paarprogrammierung ein. Dabei handelt es sich um eine agile Softwareentwicklungstechnik, bei der zwei Programmierer an einer Workstation arbeiten. Einer der Programmierer ist dabei der sogenannte Pilot (englisch „driver“) und schreibt den Code, während der andere, der sogenannte Navigator (english „navigator“), den Code prüft und gleichzeitig die Übersicht behält. Man konnte feststellen, dass zwar die Entwicklungszeit um ca. 15% höher liegt, dass bei der Paarprogrammierung dafür aber die Qualität des Designs, die technischen Fähigkeiten und die Teamkommunikation verbessert werden und gleichzeitig Defekte und Personalrisiken gemindert werden (Cockburn & Williams, 2001). Die Paarprogrammierung kam im Zuge des Extreme Programming 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).

### 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 angewendet werden. MLOps kommt vom englischen 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 Integration, Testen, Release, Bereitstellung und Infrastrukturmanagement (Google, 2020b). Befassen wir uns zunächst mit der kontinuierliche Integration, der kontinuierliche Lieferung und dem kontinuierlichen Testen.

###### Kontinuierliche Integration

Die kontinuierliche Integration (CI) ist eine Entwicklungsphilosophie, bei der Entwickler in regelmäßigen Abständen Code in einem Versionsverwaltungs-Repository durch den sogenannten Commit ablegen oder committen; dies passiert normallerweise mindestens ein Mal am Tag. Ein Anlass dafür ist, dass man dadurch Fehler und andere Softwarequalitätsprobleme leichter und an kleineren Codestücken finden kann. Das ist an großen, über längerer Zeiträume hinweg geschriebenen Codestücken sehr viel 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 regelmäßige Integration der Codebasis ja 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, Build- und Bereitstellungsskripts, 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 Mitglied eines Entwicklungsteams Bereitschaft und Disziplin mitbringen. 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-Repositorys, die für die Ausführung der Softwarekompilierung notwendigen Skripts und die automatisierten Commit-Tests festgelegt.

###### Kontinuierliches Testen

Wie im ersten Abschnitt dieser Lerneinheit erwähnt, unterstützt das automatisierte Testen die Testingenieure während des Softwareentwicklung-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 (e.g. Modul- und Funktionalitätstests) bereits in den CI-Teil der Pipeline integriert worden sein (CI weißt dann auf Probleme vor oder während der Integration hin).

Softwareentwicklung

Tests, für die Lieferungskonditionen vollständig sein müssen, z.B. Performanz- oder Sicherheitstests, sollten in die kontinuierliche Lieferung integriert sein. 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 kontinuierliche Lieferung können Unternehmen Dienstverbesserungen schnell, effizient und zuverlässig an den Markt liefern. Während der Entwicklungsphase verwendet jedes Entwicklungsteam eine oder mehrere Entwicklungs- und Testumgebungen zur Bereitstellung von Anwendungsänderungen zum Testen. Dieser Prozess kann anhand von CI/CD-Tools, wie Jenkins (Jenkins, o.D.-c), CircleCI (CircleCI, o.D.) oder Travis CI (Travis CI, o.D.) automatisiert werden.

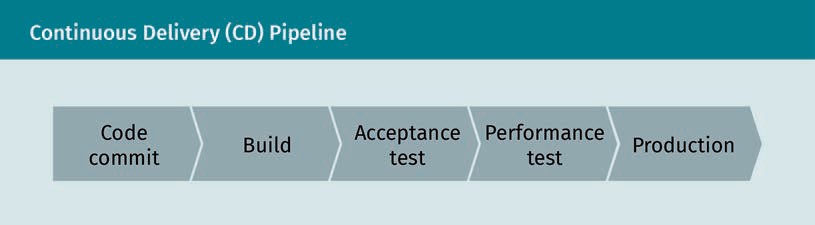
Eine typische CD-Pipeline hat folgende Schritte (Chen, 2015).

1. Commit ausführen: Beim Commit bekommen Entwickler schnelles Feedback zum bereits von ihnen eingecheckten Code. Wenn ein Entwickler 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 erhalten eine Fehlermeldung. Der Entwickler klärt das Problem und checkt den Code ein. Nach erneuter und fehlerfreier Ausführung des Codes kommt 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 Artifacts generiert. Diese Artifacts werden ins Repository geladen, in dem sie für die Bereitstellung und die Lieferung verwaltet werden. Alle der nachfolgenden Pipeline-Schritte werden mit diesen Artifacts ausgeführt.
3. Akzeptanztest: Durch diesen Schritt wird gewährleistet, dass die Software alle definierten Benutzeranforderungen 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, 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, wie Codeänderungen die Leistung der Software beeinflußt. Die Pipeline stellt eine Reihe von Performanztests zur Verfügung und erstellt nach Ausführung Berichte.
5. Produktion: Der letzte Schritt ist die Bereitstellung in der Produktionsumgebung.

Diese Schritte können anhand von CI/CD-Tools, wie Jenkins, automatisiert werden. Entwickler könnten dann Jenkins zur Beschreibung der Pipeline in einer Jenkinsfile, die verschiedene Stufen enthält, verwenden. Die Jenkinsfile enthält auch Umgebungsvariablen, Zertifikate, und andere Parameter (Jenkins, o.D.-c).

Code Coverage

Eine Erfolgsmessung der Ausführung entwickelter Software während eines Tests durch Metriken wie Funktionsabdeckung und Zeilenabdeckung.



###### Kontinuierliches Training

Kontinuierliches Training (CT) wird durch die Automatisierung der ML-Pipeline erreicht. Um die ML-Pipeline durch die Versorgung des Models 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 ob die Pipeline-Ausführung angehalten werden soll.
  + Automatische Modellvalidierung: Dieser Schritt wird mit den neuen Daten direkt nach dem Modelltraining ausgeführt. 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.

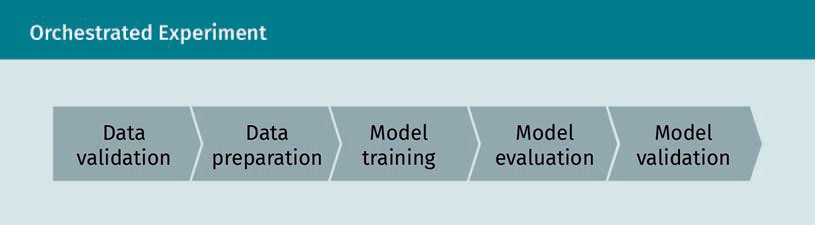
Eine schematische Abbildung einer automatisierten ML-Pipeline für kontinuierliches Training wird in Google Cloud im Anhang 1 abgebildet (Google, 2020b).

Folgende Eigenschaften sollte eine ML-Pipeline für kontinuierliches Training haben (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.
  + Kontinueirliches Training des Modells in der Produktion: Das Training des ML-Modells in der Produktion sollte automatisiert und mit frischen Daten durchgeführt werden.

Softwareentwicklung

* Symmetrie zwischen Experiment und Betrieb Ein wichtiger Aspekt einer automatisierten Pipeline ist, dass dieselbe Pipeline aus der Entwicklungs- oder Experimentumgebung in der Vorproduktion- 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.



###### MLOps und ML-Pipeline für CI/CD

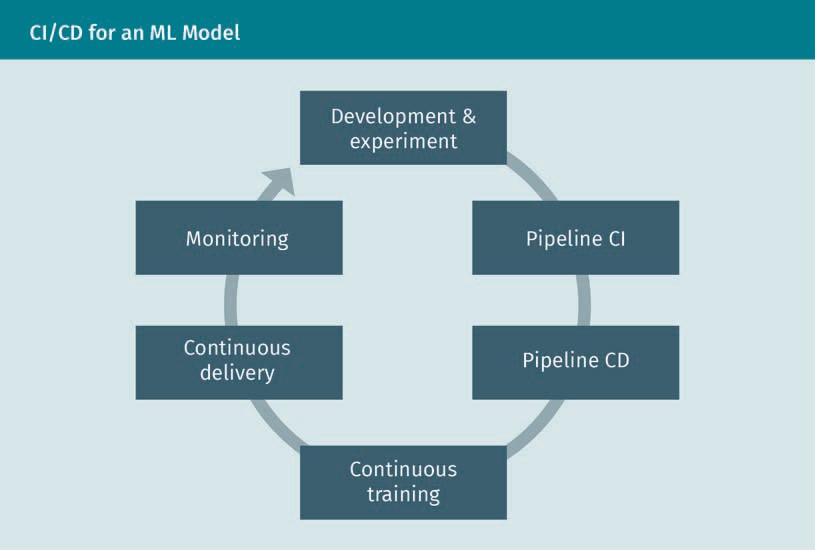
Ein ML-System ist auch ein Software-System, daher müssen ähnliche Vorgehensweisen angewendet werden, um zu gewährleisten, dass im großen Stil Builds erstellt und die ML-Systeme dann auch zuverlässig ausgeführt werden können. Es gibt jedoch eine Reihe von Unterschieden zwischen ML-Systeme und konventionellen Softwaresystemen (Google, 2020b). Einem ML-Projektteam müssen beispielsweise Datenwissenschaftler mit Erfahrung in explorativer Datenanalyse, Modellentwicklung und Experimenten angehören, sie müssen natürlich nicht gleichzeitig auch noch Softwareentwicklungsfähigkeiten haben. Auch der Entwicklungsprozess ist ein anderer, denn ML-Systementwicklung basiert auf Experimenten. Das heißt, dass eine Entwickler verschiedene Algorithmen, Modellierungstechniken und Parameter ausprobieren sollte, um die beste Lösung des Problems zu finden. Neben konventionellen Tests—wie Modul- und Integrationstests—sollten auch die Datenvalidierung und die Qualitätsvalidierung der trainierten Modelle getestet werden. Zudem muss möglicherweise auch die Bereitstellung der Ml-Systeme anders erfolgen, da manchmal eine mehrstufige Pipeline notwendig ist, wodurch die Bereitstellung komplexer wird. Da ML-Systeme sich ständig und sehr dynamisch weiterentwickeln, ist die Möglichkeit der Leistungsminderung eines Modells höher als das bei gewöhnlichen Softwaresystemen der Fall ist.

Aufgrund der oben angeführten Eigenschaften von ML-Systemen besteht auch im Rahmen der kontinuierlichen Integration und Lieferung ein Unterschied 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-Trainingpipeline automatisch auch andere Dienste (z.B. Modellvorhersage) bereitgestellt werden.
* Das kontinuierliche Training ist neue und trifft nur auf ML-Systeme zu. Dabei geht es um das automatische Training und die Pflege der Modelle.

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

1. 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 Quelle-Repository als Output dieses Schritts gepusht.
2. 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 Artifacts, die in den nächsten Schritten verwendet werden.
3. Kontinuierliche Lieferung der Pipeline: Die Artifacts aus Schritt 2 werden in der Zielumgebung bereitgestellt. Das Output aus diesem Schritt ist eine bereitgestellte Pipeline mit dem neuen ML-Modell.
4. Automatisches Auslösen: Die Pipeline wird in der Produktionsumgebung auf einen Signal oder nach einem Terminplan ausgeführt. Das Output aus diesem Schritt ist ein trainiertes Modell, das in die Modell-Registrierung gepusht wird.
5. Kontinuierliche Lieferung des Modells: Das trainierte Modell wird als Vorhersagemodell an den Kunden geliefert. Das Output aus diesem Schritt ist der Vorhersagedienst.
6. Monitoring: Statistische Daten zur Modellperformanz werden im aktiven System gesammelt. Das Output aus diesem Schritt ist ein Bericht sowie möglicherweise auch ein Signal zur Auslösung von Schritt 1 in der Pipeline.

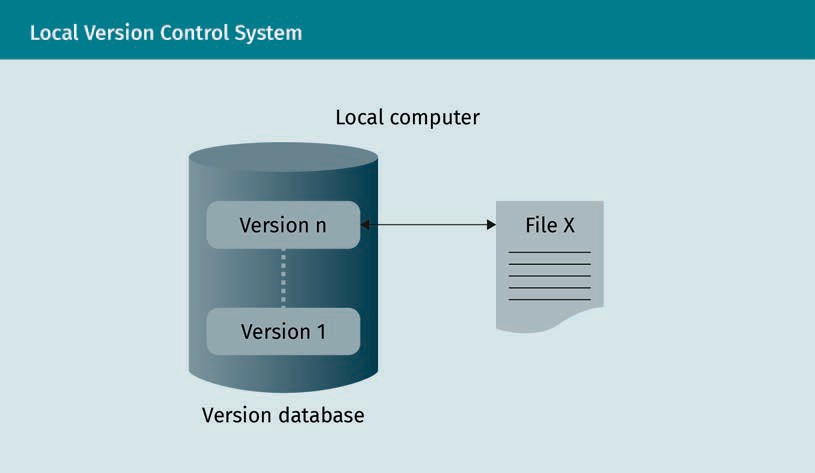


### Versionsverwaltung

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

Softwareentwicklung

Weil dies so enorm wichtig ist, wird seit Jahren an einer Lösung gearbeitet, nämlich dem Versionsverwaltungssystem (VCS). Mithilfe eines VCS können verschiedene Entwickler und Teams in einem Projekt gemeinsam und paralell an separaten Teilen der Anwendung arbeiten, während sie ein Aufzeichnungssystem führen (Humble & Farley, 2015). Mithilfe eines VCS können ausgwählte Dateien oder das gesamte Projekt in einen vorherigen (gespeicherten) Zustand gebracht werden. Das este Versionsverwaltungssystem wurde 1972 von Marc J. Rochkind von Bell Labs entwickelt und hieß SCCS (Source Code Control System) (Rochkind, 1975). SCCS folgten viele Open-Source-Versionsverwaltungssystem, z.B. RCS (Tichy, 1982), CVS (Price, 2005), Apache Subversion (Free Software Directory, o.D.), and Git (Git, o.D.) sowie viele kommerzielle Lösungen, wie Perforce (Perforce, o.D.), StarTeam (StarTeam, o.D.), IBM Rational ClearCase (IBM, o.D.-b), Mercurial (Mercurial, o.D.) und Microsoft Team Foundation Server (MS Teams, o.D.). In diesem Abschnitt geht es um eine der Open-Source-Versionsverwaltungssysteme. Da SCCS- und RCS-Lösungen heut nicht mehr sehr gebräuchlich sind, werden sie hier nicht behandelt. Der Schwerpunkt soll dagegen auf den am häufigsten eingesetzten Versionsverwaltungssystemen liegen: Git. Bevor wir näher auf Git eingehen, werfen wir einen kurzen Blick in die verschiedenen VCS-Kategorien. Beginnen wir mit den lokalen Versionsverwaltungssystemen. Diese Kategorie ist die einfachste Form eines VCS-Systems, die wahrscheinlich alle Entwickler, als sie mit dem Kodieren angefangen haben, einmal verwendet haben. Dabei werden modifizierte Dateien in ein neues, mit dem Datum und bestenfalls auch mit einem Zeitstempel versehenen Verzeichnis kopiert. Obwohl diese der einfachste Ansatz für einen einzelnen Entwickler ist, ist die Fehleranfälligkeit auch extrem hoch. Beispielsweise ist es schwer, die Übersicht und Ordnung der Directories zu behalten, aber sehr leicht, Dateien zu überschreiben. Um dieses Problem zu verhindern, kann man eine einfache lokale Versionsdatenbank, durch die Änderungen nachgehalten werden, entwickeln.

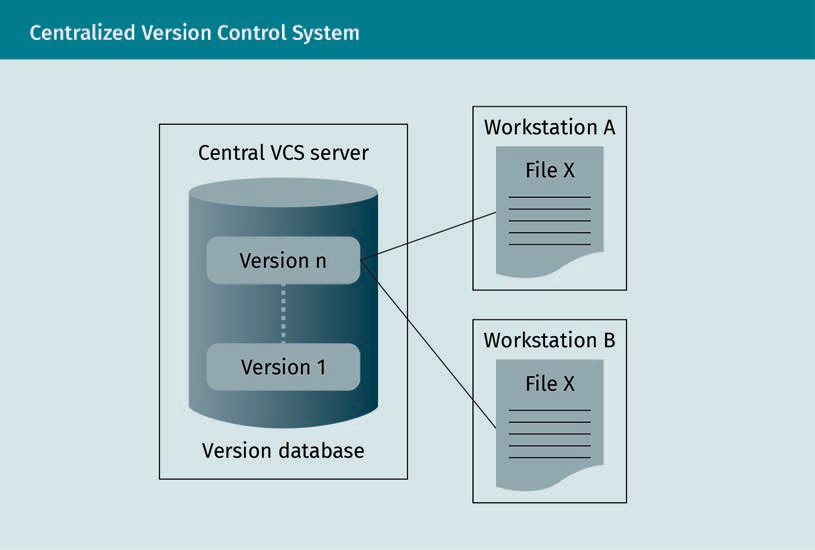


Die zweite Kategorie sind die zentralen Versionsverwaltungssysteme (CVCS). Wenn mehr als ein Entwickler oder mehr als eine Entwicklungsworkstation am Projekt beteiligt sind, sollte eine andere VCS-Lösung gewählt werden, wie in der nächsten Abbildung dargestellt. In einem CVCS werden die Informationen zu Dateiänderungen auf einem Zentralserver gespeichert und mit Clients (Workstations) geteilt. Unter den Produkten in dieser Kategorie sollten Subversion (Free Software Directory, o.D.), Perforce (Perforce, o.D.) und CSV (Price, 2005) erwähnt werden.

Single Point of Failure

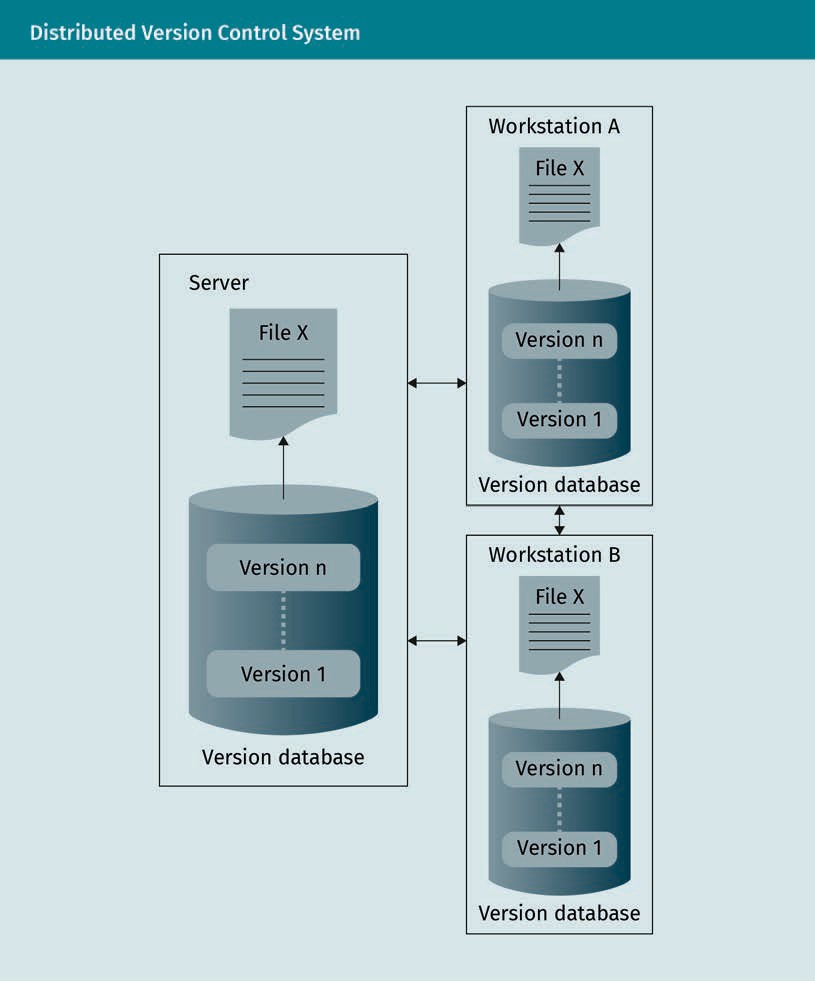
Ein nicht-redundanter Teil eines Systems, dessen Fehlfunktion das gesamte System zum Erliegen bringt, wird als SPOF oder Single Point of Failure bezeichnet (AVI Networks, o.D.).

Dieses System hat Vorteile, z.B. Können alle berechtigten Teammitglieder sehen, was andere im Team tun. Administratoren können diese Rollen festlegen. Aber es gibt auch Nachteil, z.B. Könnte ein Single Point of Failure, beispielsweise ein Server bei einem Fehler die Workstations so beeinträchtigen, dass Änderungen nicht mehr nachverfolgt werden können.



Die beste Lösung, um Problem verbunden mit lokalen und zentralen VCS zu vermeiden, ist ein verteiltes Versionsverwaltungssystem (DVCS), wie Git (Git, o.D.) oder Mercurial (Mercurial, o.D.). In einem DVCS checken Entwickler nicht die letzte Dateiversion aus, und sie erstellen eine Spiegelkopie des Repository. So wird das Single Point of Failure-Problem gelöst, da beim DVCS jedes Repository eines einzelnen Benutzers auf den Zentralserver zur Wiederherstellung der verlorenen Daten kopiert werden kann.

Softwareentwicklung



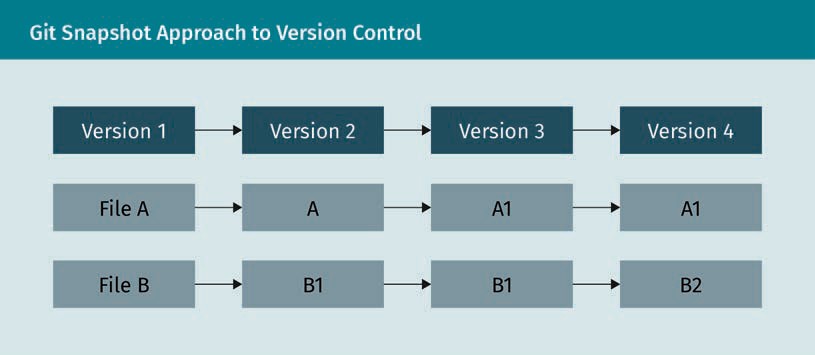
###### Git

Die Geschichte des Git-Projekts ist eng mit dem Open-Source-Softwareproject, Linux Kernel, verbunden. 2005 entschied sich das Linux Kernel-Team 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 besonders nicht-lineare Entwicklung unterstützt. Git hat tausende voll verteilte Parallelzweig, mit denen große Projekte gemanagt werden können. Während andere VCSs Änderungsinformationen als Liste von dateibasierten Änderungen, d.h. als Dateisatz mit den entsprechenden Änderungen, speichern, speichert Git eine Momentaufnahme jedes Projektzustands sofort nach einem Commit, bei dem Änderungen auf dem lokalen Repository gespeichert werden, durch den Benutzer.

EntwicklungszweigEine Kopie eines Hauptrepository des Versionsverwaltungssystems.

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

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.).



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

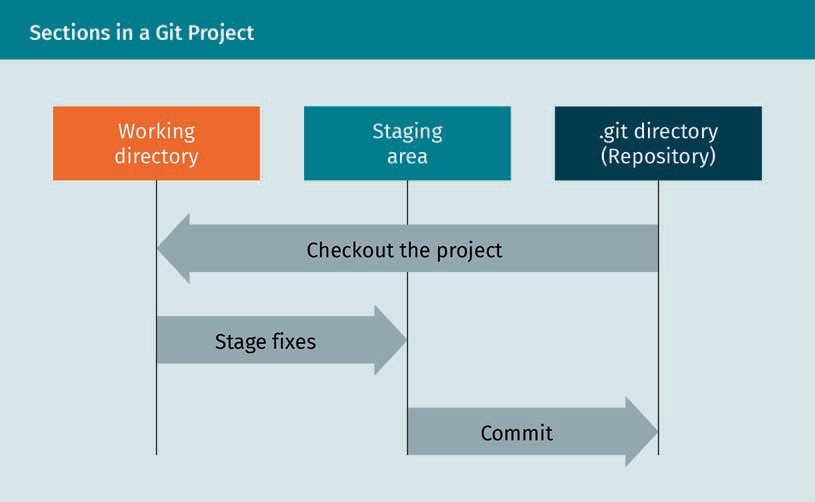
Eine Datei in Git kann drei verschiedene Zustände haben:

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 Benutzer 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.

Auf Basis dieser drei Zustände gibt es in einem Git-Projekt drei Hauptabschnitte oder -stufen. Das erste ist das Arbeitsverzeichnis, ein einzelner Checkout einer Projektversion. Die Dateien in diesem Verzeichnis werden aus dem Git-Repository entnommen, wobei Änderungen aus dem Remoterepository in den aktuellen Zweig integriert werden. Das zweite 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 Benutzer ein Repository aus einer anderen Workstation klont, heißt das, dass das Git-Repository kopiert wird. Dieser Prozess wird in der folgenden Abbildung gezeigt. Wenn man die Abschnitte eines Git-Projekts betrachtet, hat ein einfacher Git-Workflow folgende Schritte (Git, o.D.).

Softwareentwicklung

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



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

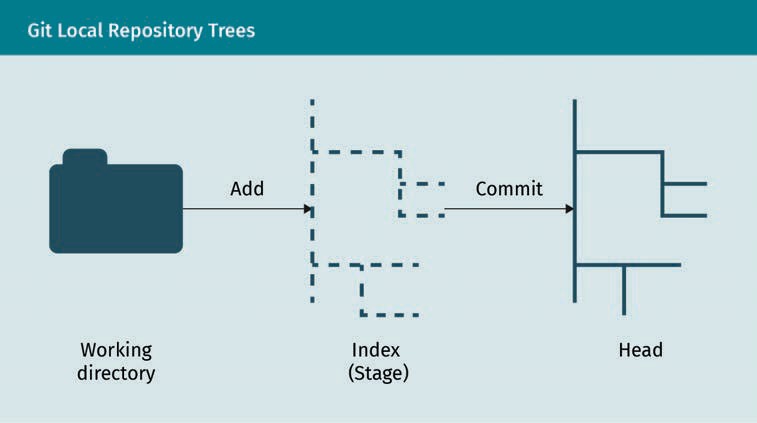
1. Ein neues Repository im Projektverzeichnis erstellen.

git init

1. Eine Arbeitskopie auf einem lokalen Repository durch die Ausführung des Befehls erstellen.

git clone /path/to/repository

1. Das lokale Repository besteht aus drei Baumstrukturen:
   1. Das Working Directory (Arbeitsverzeichnis) enthält die eigentlichen Dateien.
   2. Index ist der Staging-Bereich.
   3. HEAD zeigt auf die letzte Commit-Version.



1. Änderungen vorschlagen (dem Index hinzufügen).

git add <filename>

1. Commit der Änderungen in den Head durchführen.

> git commit -m "Commit message"

1. Änderungen ins Remoterepository schicken.

git push origin master

1. Einen neuen Entwicklungszweig („branch\_1“) erstellen.

git checkout -b branch \_1

1. Switch zum Hauptentwicklungszweig, dem Master, vornehmen.

Git checkout master

1. Push des Entwicklungszweig ins Remoterepository durchführen.

Git push origin <branch>

1. 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 ist zwischen Git und GitHub ist, dass Git ein Open-Source-Tool ist, mit dem Entwickler ihren Quellcode lokal verwalten können, während GitHub eine cloudbasierte Plattform ist, auf der Entwickler ihre Projekte mit der Community teilen können, durch die sie Zugriff auf Entwicklertools haben, wo sie gegenseitig Code überprüfen können und vieles mehr. Wie bei Git liegt auch einem GitHub-Projekt ein Projektrepository zugrunde. Ein Projektrepository enthält alle Komponenten, die für ein Projekt benötigt werden: Dateien, Ordner, Tabellenkalkulationen, Datasets usw. Normallerweise ist darin auch eine README-Datei enthalten, die Informationen über das Projekt enthält.

Softwareentwicklung

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 testen, bevor ein Commit in den master stattfindet. Wenn ein Entwickler in einem Entwicklungszweigs des master arbeitet und im master Änderungen entstehen, könnte der Entwickler die Änderungen aus dem master in den Entwicklungszweig holen.



### Entwicklertools

Programmierungs- oder Softwareentwicklungstools sind Computerprogramme, die Softwareentwickler und Entwicklungsteams bei der Durchführung von Softwareprojekten unterstützen. Diese Tools werden dazu verwendet, Quellcode für Programme mit Text-Editorer 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 debuggen, und Programme und die zugehörige Dokumentation anhand von Versionsverwaltungssystemen zu speichern und zu verwalten. In diesem Abschnitt betrachten wir einige der gebräuchlichsten Entwicklertools.

###### Befehlszeilenschnittstelle

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

Ein Befehl in einer Betriebssystem-Befehlszeilenschnittstelle hat z.B. Folgende Komponenten:

prompt command parameter\_1, …, parameter\_n

Shell

Eine Software, bei der ein Benutzer mit dem Betriebssystem oder dem Programm selbst interagiert.

Die Eingabeaufforderung (prompt) gibt den Kontext für den Benutzer (und endet gewöhnlich mit einem der folgenden Zeichen: $, %, #, :, > odrr –).

Der Befehl (command) wird vom Benutzer zur Ausführung einer bestimmten Aufgabe eingegeben, und parameters sind optionale Parameter vom Client, um den Befehl zu steueren 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, z.B. für Anfänger, zum Inline-Editing und zum Debuggen. Es gibt jedoch integrierte Tools für alle Befehlszeilenschnittstellen, die die Arbeit von Entwicklern erleichtern. Zu diesen Tools gehören der Text-Editor 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 Datenkomprimierungsprogramm Gzip(Free Software Foundation, o.D.-b).

###### Integrierte Entwicklungsumgebung (IDE)

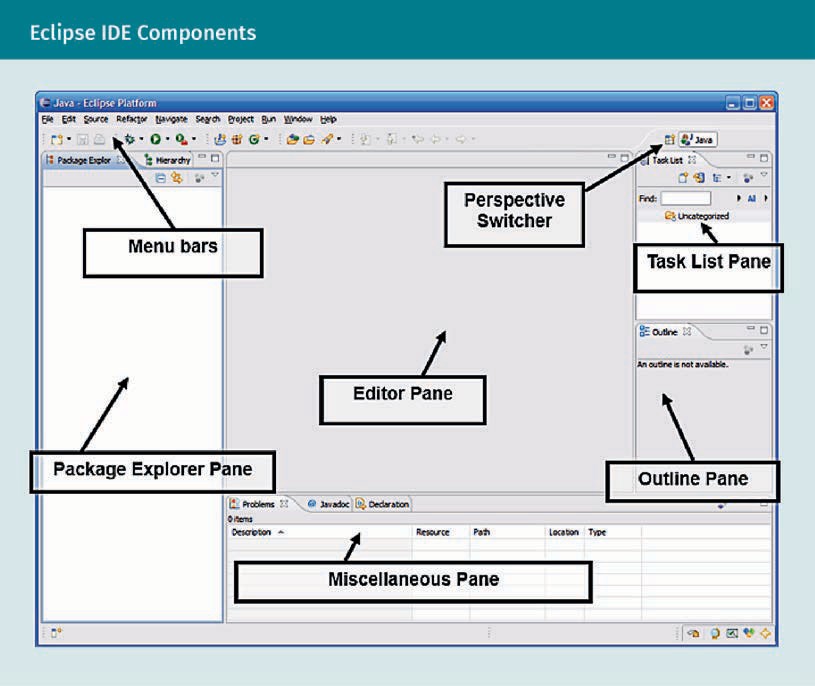
Eine integrierte Entwicklungsumgebung (IDE) ist ein einzelnes Programm oder eine Plattform, die mehrere Entwicklungsbereiche zur Softwareentwicklung umfasst. Dazu gehören Microsoft Visual Studio, Eclipse, NetBeans und PyCharm. Zu den typischen Funktionen von modernen IDEs gehören

* 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, o.D.).
* Quellcode-Editor. Der Entwickler kann die IDE zum Schreiben und editieren von Quellcode verwenden. Der Editiervorgang wird dabei von Funktionen, wie intelligente 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 hebt für die unterstützten Programmiersprachen Text in verschiedenen Farben und Schriftarten hervor, wie nachstehend abgebildet.

Softwareentwicklung



Nun befassen wir uns mit der gängigsten Open-Source-IDE: Eclipse (Eclipse, o.D.). Eclipse begann als proprietäre Technologie unter der Führung von IBM. 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 an und war ursprünglich für Java gedacht. Heute unterstützt Eclipse viele andere Sprachen, z.B. C, C++, Python und Ruby (Unversität von Maryland, 2018). Die Komponenten der Eclipse-IDE werden nachstehend dargestellt.



* 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 an Aufgaben, die fertig gestellt werden müssen
* Outline-Bereich: hierarchische Ansicht einer Quelldatei

###### Jupyter Notebook

Jupyter Notebook ist ein Internet-basiertes, 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 zur Ausführung von Python-Code verwendet, aber es unterstützt ca. 40 verschiedene Programmiersprachen. Im Allgemeinen kann es zur Datenbereinigung und -transformation, numerischen Simulation, statistischer Modellierung, Datenvisualisierung, Maschinelles Lernen und vielem mehr verwendet werden. Jupyter Notebook vereint zwei Funktionen. Zum einen enthält es eine Webanwendung (oder ein Browser-basiertes Tool) zur interaktiven Verwaltung von Dokumenten, z.B. beschreibendem Text, Mathematik, Berechnungen und die zugehörigen grafischen Medien, wie Diagramme oder 3D-Visualisierungen.

Softwareentwicklung

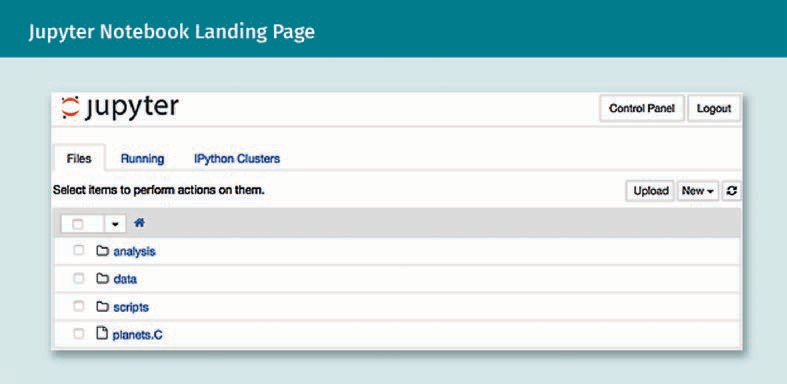
Die Webanwendung ermöglicht das Editieren von Texten in einem Browser mit automatischer Syntaxhervorhebung, die Ausführung von Code aus dem Browser, das Darstellen der Berechnungsergebnisse in PNG, SVG, HTML und anderen Rich Media-Formaten, und dabei werden auch mathematische Notationen mit LaTeX nicht 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, Images 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.).

Einsatz von Jupyter Notebook

1. Ein Notebook-Server kann von der Befehlszeile mit dem folgenden Befehl gestartet werden (Jupyter Team, o.D.):

>jupyter notebook

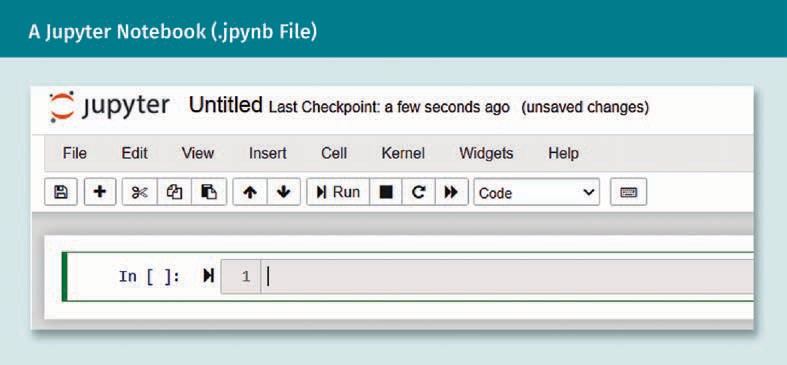
1. Die Landingpage der Jupyter Notebook-Webanwendung ist standardmäßig http:// 127.0.0.1:8888 und wird als Dashboard bezeichnet. Sie enthält die im Notizbuch-Verzeichnis enthaltenen Notizbücher.

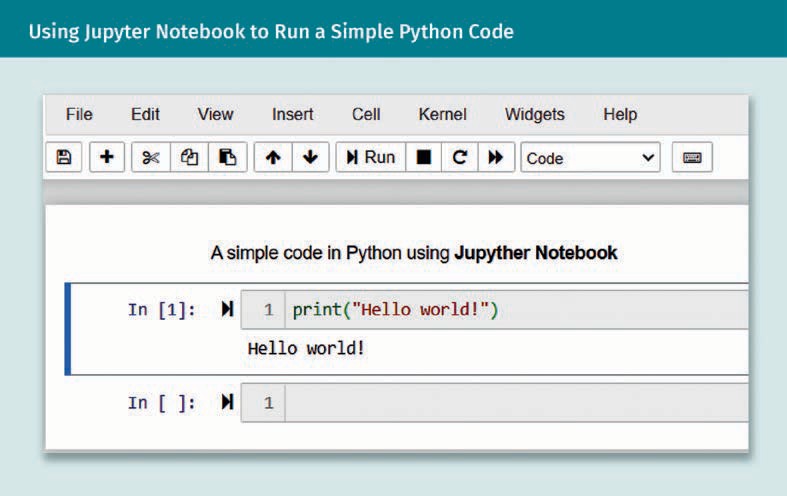


1. Im Dashboard lassen sich unter Neu/Notizbücher neue Notizbücher erstellen. Das neue Notizbuch, in der Form einer .ipynb-Datei, wird unten abgebildet.

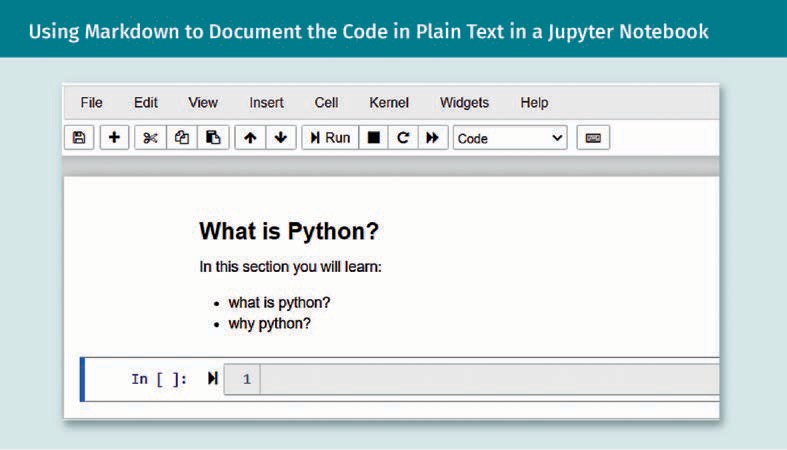
JSON-Dateien

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

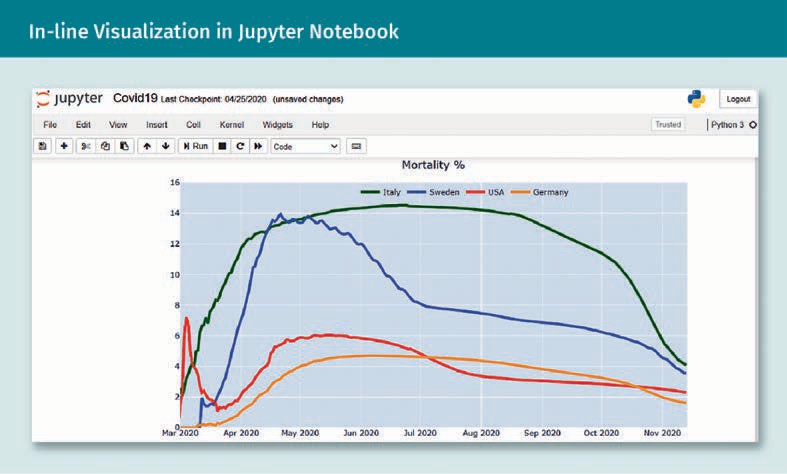


1. Danach kann ein Benutzer im Notizbuch codieren.
2. Code kann in natürlicher Sprache mit Markdown dokumentiert werden. Wie die folgende Abbildung zeigt, wird auch Textmarkup (Kursiv, Fett, Aufzählungen usw.) unterstützt.

Softwareentwicklung



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



Zusammenfassung

In dieser Lerneinheit wurden die verschiedenen Ansätze zur Softwareentwicklung behandelt, wobei besondere Betonung auf das Testen gelegt wurde. Zunächst beschäftigten wir uns mit verschiedenen Testszenarien, wie Modul-, Integrations-, System-, funktional und nicht-funktionale Tests nach der Kategorien im Testquadranten von Marick. Danach wurde besprochen, wie ein Machine-Learning-System, als nicht-deterministisches Modelle, 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: testgetriebene Entwicklung (TDD), verhaltensgetriebene Entwicklung (BDD) und akzeptanztest-getriebene Entwicklung (ATDD).

Schließlich befassten wir uns mit der Automatisierung on 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. Danach wurden die Prinzipien von Versionsverwaltungssystemen für Softwareprojekte und zwei der gebräuchlichsten Lösungen vorgestellt: Git und GitHub. Und schließlich gingen wir auf Entwicklertools, wie Befehlszeilenschnittstelle und integrierte Entwicklungsumgebung (IDE), ein.



# Lerneinheit 4

## API

#### LERNZIELE

Nach Abschluss dieser Lerneinheit verstehen Sie ...

... die verschiedenen Interaktionsarten zwischen Software und Diensten.

... die wichtigsten Prinzipien für das Design und die Erstellung von Programmierschnittstellen.

... wie sich gutes von schlechtem Schnittstellendesign unterscheidet.

... wie Bibliotheken in Python mit gutem Design erstellt werden.

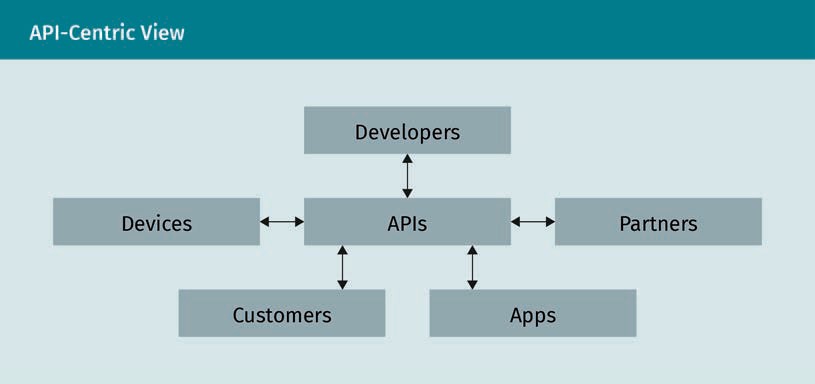
DL-E-DLMDSSEDIS01-U04

1. API

### Einführung

Aufgrund der wachsenden Digitalisierung und Vernetzung haben viele Menschen Zugriff auf intelligente Geräte, die sie für eine zunehmende Zahl und Art von Anwendungen verwenden. Das heißt, dass selbst technisch unerfahrene bzw. unbewandte Benutzer regelmäßig komplexe Software verwenden müssen. Wenn das gelingen soll, muss der Interaktion zwischen Mensch und Computer gutes und leicht verständliches Design zugrunde liegen. Darüber hinaus muss jede Schnittstelle zu einer Software stabil und zuverlässig sein, so dass die Dienste, die wir in Anspruch nehmen wollen, reibungslos funktionieren und nicht plötzlich abbrechen. Nehmen wir das Einzelhandelsgeschäft von Amazon als Beispiel. E-Commerce-Plattformen, wie die von Amazon, kennt heute jeder, und jeder hat dort schon einmal nach Produkten gesucht, welche ausgewählt und in den Einkaufskorb gelegt. Die Website, auf der wir uns intuitiv bewegen, ist nur die Benutzeroberfläche eines riesigen und komplizierten Software- und Hardware-Backends, das von Zehntausenden von Maschinen auf der ganzen Welt und unzähligen Ingenieuren, die diese Plattform in den letzten zwei Jahrzehnten aufgebaut haben, betrieben wird. Da wir diese Dienste und Anwendungen heute tagtäglich benutzen, vergisst man schnell, wie groß die Diskrepanz zwischen dem, was wir sehen, und dem Rest oft sein kann. Die Interaktion eines Benutzers mit der Anwendung ist wie die Spitze des Eisbergs der Komplexität. Manchmal kann allein die Benutzeroberfläche schon überwältigen oder einfach kein gutes Design haben. Aber auf einer Benutzeroberfläche ist nicht nur das wichtig, was ein Benutzer sieht, sondern auch das, was eben nicht gezeigt werden soll. Mit dem Klicken auf einer Website fängt es allerdings erst an. Softwareingenieure und Datenwissenschaftler arbeiten täglich am Computer, wobei ihre Interaktion mit den Systemen eindeutig anders aussieht als beim Durchschnittsnutzer, dennoch müssen auch sie sich auf die Verträge mit Programmen und Tools verlassen können. Diese Verträge sind die so genannten Anwendungsprogrammierschnittstellen oder kurz APIs. Auch Hobbyprogrammierer kennen heute den Ausdruck API. Kommen wir aufs Beispiel Amazon zurück. Die Veröffentlichung des Technologie-Stacks über APIs als Amazon Web Services (AWS) ist inzwischen ein genauso großes Geschäft wie der Einzelhandel von Amazon. Es ist zwar nicht jeder Klick auf einer Website ein API-Aufruf, aber jede Geschäftstransaktion eines Benutzers entspricht einem API-Aufruf. Es lässt sich durchaus eine API-zentrische Ansicht der modernen, digitalisierten Geschäftstransaktionen vertreten, bei der nicht nur API-Entwickler, sondern auch Anwendungen und Geräte, Kunden und strategische Geschäftspartner über APIs interagieren.

API



Diese Lerneinheit bietet einen Überblick der geläufigen API-Beispiele, also über die vielen Ansätze zur produktiven Interaktion mit Computern. Dabei konzentrieren wir uns hier auf die spezielle Benutzergruppe der Ingenieure und Wissenschaftler, also Experten wie Teilnehmer an diesem Kurs. Abgesehen von dem Verständnis, was für ein Computerprogramm und wie es verwendet wird, müssen erfahrene Profis lernen, Schnittstellen kritisch zu betrachten. Denn am Ende des Programmierens steht eine Schnittstelle für einen Benutzer. Daher ist es lohnenswert, sich die Fähigkeiten, die für gute Schnittstellen notwendig sind, zu erarbeiten. Die Entwicklung und die Verwendung 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 Fowler zu sagen „Jeder Idiot kann Code für einen Computer schreiben. Gute Programmierer schreiben Code, den Menschen verstehen können“ (Fowler & Beck, 1999, S. 15).

### Interaktion mit Software und Diensten

Moderne Computer sind äußerst komplex. Selbst als Experte kann man bei der Interaktion mit dem Computer nur deshalb produktiv sein, weil es viele Abstraktionsebenen gibt, die geschickt verstecken, was im Hintergrund passiert. Macht man zum Beispiel eine Analyse in Python, sollte man sich auf den speziellen Anwendungsfall konzentrieren und nicht darauf, wie Code in Nullen und Einsen übersetzt wird oder darauf, wie der CPU vom Betriebssystem zur Analyseausführung eingesetzt wird. Letztendlich muss man sich darauf verlassen, dass der Code, also der kleine Teil aller Programmierfunktionen, genau das tut, was man möchte. Das heißt, als Entwickler verwendet man ständig Anwendungsprogrammierschnittstellen, um Komplexität zu vermeiden und produktiv zu sein. Wie bereits in der Einführung 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 von ihr erstellten Formate klar darlegen und
    - kommunizieren, welche Konventionen befolgt werden und was das gewollte Verhalten jeder Anfrage ist.

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

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, das in der Python-Umgebung installiert ist, sein. 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:

ModuleNotFoundError: No module named ' testfailure '

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

###### Geschichte und Entwicklung von APIs

Die Entwicklung des Terminus API seit ihrer Einführung ist interessant. Aus heutiger Sicht haben Programmierer schon mit den ersten Programmen APIs entwickelt. Ursprünglich wurde die Benennung API nur für Anwendungsschnittstellen zum Benutzer verwendet, allerdings wurde die Bedeutung schnell erweitert. Die Benennung selbst wurde in den 60er Jahre zum ersten Mal formell erwähnt (Cotton & Greatorex, 1968), aber es dauerte noch etwas bis der Begriff populär wurde. Cotton und Greatorex (1968) haben eine gleich bleibende Anwendungsschnittstelle in Fortran vorgeschlagen, 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 offensichtlich, aber in den Anfangstagen der Programmierung war das kein leichtes Ziel. 1974 dehnte sich der Begriff API auf Datenbankverwaltungssysteme aus, aber die Anwendungsschnittstelle war streng von den anderen Interaktionsarten mit der Datenbank, z.B. die Suche, getrennt (Date, 2019). Dies stellte sich als zu rigoros heraus, und folgende Generationen stellten fest, dass sich durch die Erweiterung des Begriffs Anwendungsschnittstelle alle möglichen Interaktionen vereinen ließen.

API

Und so wurde die Entwicklung von differenzierten, integrierten Schnittstellen die Norm in allen Bereichen der Informatik, nicht nur beim Datenbankdesign (Berg et al., 1981). In den 90er Jahren war allgemein anerkannt, dass die Benennung API einen weiteren Begriff in allen Bereichen der Programmierung abdecken würde. Die damalige Definition war „eine Reihe von Diensten, die Programmierern zu bestimmten Aufgaben zur Verfügung steht“ (Malamud, 1990, S. 294). Mit Einführung des Internets wurden Web-APIs zu den erweiterten „netzwerkbasierten Anwendungsprogrammierschnittstellen“ (Fielding, 2000). Web-APIs sind heute so weit verbreitet, dass „API“, was historisch eigentlich einen engeren Begriff abdeckte, als Abkürzung für Web-API und die Kommunikationsprotokolle verwendet wird. Und tatsächlich wird API häufig für die noch engere Bedeutung von JSON- oder XML-basierten Web-APIs verwendet. In diesem Kapitel 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 Benutzern auf der ganzen Welt führte zu einer Explosion an Web-APIs, die sicherlich weitergehen wird. Ihr Anfang war jedoch einfach. Zu den Pionieren im kommerziellen Einsatz von Web-APIs gehörten einige Größen der Dotcomwelle, wie Salesforce, Ebay und Amazon. 2002 hat Amazon beispielsweise die Amazon Web Services (AWS) ins Leben gerufen. Mit diesem Service können Entwickler Inhalt von Amazon in ihre eigenen Websiten einbetten. AWS hat sich seitdem rasant weiterentwickelt, und die Cloud-Angebote von Amazon stehen fast auf dem Niveau des Vertriebsgeschäfts von Amazon. Mit der nächsten Welle der Web-API, die ca. 2004 begann, kam die Integration der sozialen Medien durch Tech-Giganten wie Facebook, Twitter oder Flickr. Die Entwickler-API von Twitter wurde 2006 veröffentlicht, um Ingenieuren Zugriff auf alle möglichen Daten auf der Plattform zu geben. Im selben Jahr wurde die API von Facebook gestartet, mit der Entwickler Zugang zu Informationen von Facebook-Freunden, Bildern, Beiträge usw. bekamen. Mit diesem Zug wurde die Plattform sehr attraktiv für Werbefirmen und verhalf Facebook dazu, weltweit Macht in der Branche der soziale Medien zu erlangen. In einer weiteren Welle begannen Unternehmen, ihre Infrastruktur und Dienste in die Cloud zu verlagern. Als Pionier muss Amazon genannt werden, das 2006 mit einer Reihe neuer Dienste, angefangen mit Amazon Simple Storage Service (S3) und Amazon Elastic Compute Cloud (EC2), auf den Markt kam. Beide Dienste tragen heute noch zum Erfolg von AWS bei. Während Amazon S3 Cloud-Speicher mit einfacher nutzungsbasierter Bezahlung erschlossen hat, machte es Amazon EC2 Entwicklern mögliche, 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 zugeschrieben wird, ist bemerkenswert (API Evangelist, 2012). Bis zum heutigen Tag müssen Teams bei Amazon ihre Funktionalität anderen Teams als Dienst zur Verfügung stellen. Obwohl die Philosophie kurzfristig einiges gekostet haben dürfte, denn APIs zu bauen ist nicht billig, 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. Ganz kurz müssen wir noch auf eine weitere bemerkenswerte Veränderung in der Tech-Landschaft, die eine Auswirkung auf APIs hatte, eingehen. Mit dem Aufkommen des Internet der Dinge sind immer mehr Geräte mit dem Internet verbunden und kommen daher auch für APIs in Frage. Zu den Beispielen gehört Alexa von Amazon (2014), das API-Zugriff für alle wichtigen Funktionen, wie Sprachsteuerung, Weckfunktion oder Wetterdienst, ermöglicht. Fitbit (2017) stellt API-unterstützte Wearables zum Messen von Fitnessdaten her. Im historischen Kontext spricht nichts mehr für die Wichtigkeit von APIs und das kommerzielle Interesse, 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). Abschießend lässt sich feststellen, dass APIs und ihre Benutzung heute allgegenwärtig sind und 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 übers Internet zugrunde.
    - Sie sind der Eckpfeiler des Internet der Dinge und verbinden intelligente Geräte, wie Autos, Kühlschränke oder Staubsauger.

###### Spezifikationen und Verträge

Jede Software hat einen Anbieter, Unternehmen oder Teams, die sie geschrieben und gebaut haben und die sie derzeit warten, und Verbraucher, die die Software für ihre Bedürfnisse nutzen. Verbraucher können einerseits Menschen und andererseits andere Maschinen sein, die die Software nutzen. Eine API kann, ganz gleich welche tatsächliche Rolle sie spielt, als Spezifikation oder Vertrag zwischen dem Anbieter und dem Verbraucher betrachtet werden. Wie bei anderen Verträgen müssen sich die Parteien zunächst darüber einigen, um was es beim Vertrag geht, und dies dann festhalten. Im Software-Geschäft kann der Anbieter bis zu einem gewissen Grade diktieren, wie eine Schnittstelle aussieht, muss jedoch aufpassen, dass er die Kunden nicht verliert, wenn er beispielsweise an schlechtem Design festhält. Wenn sich die Parteien auf eine API, die wie ein Vertrag ist, einigen, müssen sie sich dran halten. Firmen, die Software as a Service (SaaS) anbieten, haben meistens Service-Level-Vereinbarungen (SLAs), in denen API-Stabilität aufgeführt ist. Genauer gesagt kann eine Firma die Struktur von API-Aufrufen nicht von heute auf morgen ändern. Das führt zu bestimmten Problemen in Bezug auf Wartung und Erweiterbarkeit der APIs. Ähnlich wie Softwarepakete oder Betriebssysteme hat jede API eine Version. Die Entwicklerplattform GitHub hat eine Entwickler-API, deren URL folgendermaßen lautet

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 aber die dritte Hauptversion der API. Die ersten beiden sind veraltet und wurden vor Jahren entfernt. Die Versionierung von APIs hat viele Vorteile:

* + - In einer Version garantiert der Anbieter, dass jeder einzelne API-Aufruf eine stabile Schnittstelle hat. Die Benutzer der API, die nachgeschaltete Anwendungen entwickeln, können sich darauf verlassen, dass sie nicht plötzlich ohne Vorwarnung ausfällt.
    - Neue Funktionalität kann einer API-Version hinzugefügt werden, wenn sie nicht von alten Anforderungen abhängt. Bei Software würde man von Nebenversionen oder Software-Patches sprechen. Obwohl 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 sollten versuchen, Schnittstellen gleich beim ersten Mal richtig zu programmieren, aber das ist natürlich 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 Anbietern, mehrere Versionen der API gleichzeitig zu veröffentlichen und Verbrauchern genügend Vorbereitungszeit zu geben, so dass sie planen können, wann und wie sie auf die neueste API-Version umsteigen. Vorbereitungszeit ist für Benutzer extrem wichtig, und plötzliche Änderungen sollten vermieden werden.

API

* Ältere API-Versionen zu pflegen und gleichzeitig die Benutzer über die Einstellung zu informieren, ist zusätzliche Arbeit für die Anbieter. Aber Verbraucher kommt es 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 betrachten Dokumentation oft als ungeliebtes, notwendiges Übel, da sie sich lieber auf das Codieren konzentrieren. Sie ist jedoch zur Einhaltung eines gemeinsamen Verständnisses einer API unabdingbar. Ganz gleich wie einfach man sich eine Schnittstelle vorstellt, sie muss schlussendlich von einem Menschen bedienbar sein, selbst wenn es nur darum geht, einer anderen Maschine Anweisungen zu geben. Und dabei besteht immer das Risiko von Missverständnissen. Manchmal wird behauptet, dass die API ja die Dokumentation und der Code deren Implementierung ist. Aus der Perspektive der Softwareplanung ist das durchaus verständlich. Vor dem Manifest für Agile Softwareentwicklung war es in der Informatik gebräuchlich, explizite und äußerst detaillierte Spezifikationen (kurz Specs) zur schreiben, an die man sich halten musste, bevor mit der Codierung oder der Produktion begonnen werden konnte. Eine Spezifikation - für eine API oder ein anderes Produkt - kann als Ergebnis einer möglicherweise langen Verhandlungsphase angesehen werden. Beim Projektmanagement wird dieser Prozess oft als Anforderungsentwicklung beschrieben. Aus praktischer Sicht ist Dokumentation nicht zu unterschätzen. Wenn man Murphys Gesetzt auf APIs anwenden würde, könnte man sagen, dass API-Aufrufe, die missverstanden werden können, missbraucht werden. Daher sollte man bei der Erstellung und Dokumentation von Schnittstellen so klar und präzise wie möglich sein

###### Sicherheit, Governance und Release-Richtlinien

Anhand von APIs können Unternehmen Technologien, die extern entstanden sind, im Haus integrieren, wobei Abhängigkeiten entstehen. Genauso hat ein Unternehmen, das Zugriff auf eine API anbietet, plötzlich abhängige Kunden. Beides bringt Governance-Probleme, aber auch potenzielle Sicherheitsrisiken mit sich. In Bezug auf Governance kann man sagen: Wenn sich ein Geschäft auf die Verfügbarkeit einer API verlässt und die API wird langsamer, wird auch Arbeit weniger schnell erledigt. Die Leistungsüberwachung der externen API wird für interne Geschäftsprozesse bedeutsam. Wenn ein integriertes Tool plötzlich nicht mehr funktioniert, muss zudem auch noch schnell analysiert werden, ob der Fehler aufgrund einer Änderung im Haus oder beim externen Anbieter aufgetreten ist. Der interne Entwicklungszyklus muss deshalb den externen API-Entwicklungszyklus, von dem er abhängt, miteinbeziehen. Es kommt nicht selten vor, dass große Firmen dedizierte API-Manager haben, die innerhalb der Firma alle Themen rund um die externen APIs managen, z.B. Einhaltung von unternehmenseigenen oder nationalen Normen durch die integrierten Tools. Wenn dagegen ein API öffentlich zugänglich gemacht wird, kann es vorkommen, dass Produktionsdaten des Unternehmens zugreifbar und dadurch auch angreifbar werden. Die meisten Datenanbieter im Gesundheitswesen müssen in vielen Ländern ihre Daten unbedingt in anonymisierter Form speichern. Das Sicherheitsrisiko von Unternehmen mit vertraulichen Daten kann sehr hoch sein. Daher lohnt es sich, in ausreichende Sicherheitsstandards zu investieren und das richtige Einrichten von Zugriffsrechten für die API ist dringend erforderlich. Grob gesagt gibt es drei Hauptrichtlinien, nach denen APIs freigegeben werden können (Boyd, 2014).

Veraltete Anwendungen

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

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“.

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.

* Privat: Die API wird nur firmenintern verwendet.
* Partner: Die API wird nur einer bestimmten Benutzergruppe, d.h. Geschäftspartnern, zur Verfügung gestellt.
* Öffentlich: Die API ist öffentlich verfügbar. In typisierten Programmiersprachen wird das Stichwort „öffentlich“ oft für Klassen oder Funktionen verwendet. Das heißt, dass dieser Teil der entsprechenden Bibliothek allen Programmierern, die mit dieser Bibliothek arbeiten, zugänglich ist. Dies Art „öffentlicher“ Funktionalität kann dennoch Teil einer privaten, also innerhalb einer Firma verfügbaren API sein. „Öffentlich“ heißt auch nicht umsonst oder für jeden verfügbar. Es bedeutet, dass diejenigen, die Zugriff haben und bezahlen, zur Verwendung berechtigt sind (siehe Fowler, 2002).

Einer der bereits behandelten Vorteile von APIs ist, dass die Schnittstelle innerhalb eines API-Releases stabil ist. API-Designern markieren ihre Aufrufe gewöhnlicherweise als instabil, womit sie sich damit etwas mehr Flexibilität schaffen. In der Guava Bibliothek von Google Java werden instabile Komponenten mit @Beta annotiert, womit niedrige Zuverlässigkeit der Komponente angezeigt wird, da sie sich ändern oder ganz entfernt werden könnte.

###### API-Arten

Nachdem wir Beispiele, die Geschichte von APIs und einige der wichtigsten Aspekte, die APIs ausmachen, besprochen haben, betrachten wir Ebenen und Arten von APIs. Dabei bewegen wir uns sozusagen von innen nach außen und fangen mit den Komponenten auf niedrigeren Ebenen eines Computers an, um dann auf die höheren Ebenen des Anwendungsdesigns einzugehen. Die zwiebelartigen Schichten der API-Abstraktionen ermöglichen es uns, effizient mit einem Computer zu arbeiten. Das Skript, das Sie gerade lesen, wurde in einem Textverarbeitungssystem geschrieben, ohne dass dabei auch nur eine Sekunde an die Hardware, auf der es läuft, gedacht wurde.

Betriebssysteme

Einfach ausgedrückt ist ein Betriebssystem Software, die Hardwarekomponenten und Softwareressourcen managt und das Zusammenspiel regelt. Es stellt auch herkömmliche Dienste der Systemebene, die die Benutzer brauchen, zur Verfügung. Die POSIX-Norm ging in den 80er Jahren aus der Arbeit von Richard Stallman hervor und vereinheitlichte verschiedene UNIX-Systeme unter einem Dach. Diese Norm ist die Basis des Betriebssystems Linux (Stallman, 2011). In ihr werden eine Reihe von APIs festgelegt, die in POSIX-konformen Betriebssystemen implementiert sein müssen, so dass kompilierte, auf diesen APIs beruhende Programme auf anderen POSIX-Systemen ausgeführt werden können. Ein Linux-Benutzer trifft 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, aber das geht über den Umfang dieses Kurses hinaus. Folgende Beispiele sind ebenfalls bemerkenswerte APIs, die eng mit der Betriebssystemebene zusammenarbeiten und für datenintensive Wissenschaften wichtig sind.

API

* + 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 1992 eingeführte Norm ist heute in der Obhut des gemeinnützigen Konsortium Khronos Group.
  + 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 beendet 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.

Befehlszeilenschnittstelle

Regelmäßige Nutzer 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 Anweisungen Zeile für Zeile ausführen. Unter den vielen CLI-Tools, die auf den verschiedenen Betriebssystemen zur Verfügung stehen, müssen zwei für Python Entwickler 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 mithilfe der Python-CLI ausgedruckt:

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, wie das altbekannte easy\_install, das heute populäre Poetry oder conda, das mit allen Extras ausgestattet ist, gibt).

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

pip install pandas

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

pip install --upgrade pip

Shell

Ein Befehlszeileninterpreter, 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 ist.

Es sollte vielleicht angemerkt werden, dass Entwickler 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 Zugriff über eine Programmiersprache, während „CLI“ der Teil der API ist, auf den über die Shell Zugriff entsteht.

Programmiersprachen

Nun haben wir gesehen, wie man Python als CLI nutzen kann, um Befehle in der Shell auszuführen. Aber natürlich ist Python eine volle Programmiersprache, die allgemein für das elegante Design und die hoch entwickelte Usability gepriesen werden sollte. Hier gehen wir nicht weiter auf Einzelheiten der Sprache Python ein, wollen aber anmerken, dass alle Datenarten, Variablen, integrierten Funktionen und Klassen dieser Sprache tatsächlich selbst APIs zur Benutzerinteraktion sind. Vor allem Anfängern ist es weniger bekannt, dass die Python-API viele verschiedene Implementierungen hat. Wenn ein Entwickler sagt, dass er in Python programmiert, bezieht er sich vermutlich auf CPython, eine Open-Source-Implementierung der Python-API, die auf der Programmiersprache C basiert und weit verbreitet und umsonst verfügbar ist. Es gibt jedoch auch Alternativen, wie Jython (in Java), PyPy (in RPython) und IronPython (in C#). Man darf also nicht vergessen, wenn man in einer bestimmten Programmiersprache arbeitet, dass sie alle Implementierungsformen einer in einer anderen Programmiersprache geschriebenen Designspezifikation ist. Python ist eine eindrucksvolle Sprache, aber schlussendlich handelt es sich dabei nur um eine API für Code auf niedriger Ebene. Dieses Wissen lässt sich auch gut anwenden, da man zum Beispiel in Cpython Erweiterungen von Python in C schreiben kann, und das Ergebnis könnte sogar schneller sein als gewöhnliches Python. 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 eine weitere Schicht nach außen. Die in der Praxis verwendeten Bibliotheksprogramme und Frameworks - zum Beispiel Bibliotheken für Datenanalyse oder -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 nur der konzeptionell verstehen, was diese Schnittstelle ist, die NumPy zur Verfügung stellt. Anders gesagt ist die Schnittstelle von den Implementierungsdetails getrennt. Dieses wichtige Grundprinzip aller APIs untersuchen wir im nächsten Abschnitt weiter. Davor noch ein einfaches Beispiel: 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:

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

Wir erhalten das erwartete Ergebnis

[[2. 2. 2.]

[2. 2. 2.]

[2. 2. 2.]]

API

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. Man muss nicht wissen, wie Matrizen in NumPy gespeichert werden oder wir Matrizenadditionen oder andere Operationen ausgeführt werden. Dadurch wird die kognitive Beanspruchung gemindert, und man kann sich auf den Anwendungsfall und die API, nicht die Implementierung, konzentrieren. Hochspezialisierte Bibliotheksprogramme oder Frameworks werden manchmal als domänenspezifische Sprachen (DSL) bezeichnet, wenn sie gewissermaßen De-facto-Norm und das gesamte Toolset für eine bestimmte Domäne sind. NumPy ist eine DSL für n-dimensionale Arrays in Python. pandas ist dagegen ein Beispiel einer DSL für Datenrahmen in Python. Wenn man ein Experte in datenintensiven Wissenschaften werden möchte, 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 ein Benutzer umgehen muss, und jedes derartige System hat eine Sprache bzw. APIs, die genau das tun. Moderne, in den letzten beiden Jahrzehnten erstellte Datenbanken 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 Benutzer verborgen. Relationale Datenbanken, ihre Datenstrukturen und Algorithmen zur Datenspeicherung und zum Datenabruf, können auf die verschiedensten Arten und Weisen implementiert werden. Für den Benutzer sieht das jedoch alles aus wie eine Black Box. Der Datenzugriff spielt sich nur über die domänenspezifische Sprache SQL ab.

Remoteprozeduraufrufe

Bisher haben wir uns nur mit APIs, die sich auf einen einzelnen Prozess auf einem einzelnen Computer beziehen, befasst. Es ist jedoch die Regel, dass praktisch ständig mehrere Prozesse auf einem Computer laufen. Dazu müssen diese Prozesse sehr häufig kommunizieren: Bei der gleichzeitigen Ausführung eines Textverarbeitungsprogramms zur Protokollierung einer Besprechung und deren Leitung anhand einer Konferenzsoftware ist das Betriebssystem zum Beispiel intelligent genug, eine Notiz aus dem Textverarbeitungsprogramm in den Chat der Videokonferenz zu kopieren. Das ist eine Form der prozessübergreifenden Kommunikation (IPC). Wenn wir noch einen Schritte weiter gehen, sehen wir, dass ein Kollege an einem ganz anderen Computer die Notiz in der eigenen Instanz desselben Videokonferenztools sieht, das als Prozess auf seinem Computer ausgeführt wird. Wenn Computer so miteinander kommunizieren, ist das kein einfacher Vorgang und beruht normalerweise auf komplizierten Kommunikationsprotokollen. Ein wichtiger Teil der Kommunikation zwischen mehreren Computern ist das Networking, das heißt 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 Remoteprozeduraufruf (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 muss ebenfalls wieder vom Server zum Client übertragen werden. Das heißt, dass all Objekte im Arbeitsspeicher des Client, 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 ist für die meisten modernen Web-APIs von großer Bedeutung.

REST

Der Begriff des Representational State Transfer (REST) wurde 2000 in einer Dissertation eingeführt und legt eine Reihe von Architektur-Constraints für Webanwendungen, wie Client-Server-Architektur oder zustandslose Dienste, fest. Anhand dieser Constraints können Entwickler von Webanwendungen gemeinsame Kommunikations-kanäle für das riesige, verteilte Netzwerk, das Internet, finden.

Web-Schnittstellen

Wenn man betrachtet, wie reich und vielfältig moderne Webanwendungen und mobile Anwendungen sein können, ist es vielleicht überraschend, dass fast die gesamte Kommunikation über Protokolle wie HTTP läuft. Dieses Protokoll wurde offiziell 1997 eingeführt. HTTP/2 gibt es seit 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 Evolution des HTTP-Protokolls als API ist an sich schon interessant, aber hier bleiben wir bei der Semantik von HTTP und seiner Rolle in der Praxis. Im Zentrum von HTTP stehen die Uniform Resource Identifiers (URIs), mit denen man Ressourcen auf dem Internet eindeutig identifizieren kann, und Operationen oder Methoden, die HTTP zur genauen Verarbeitung dieser Ressourcen nutzt. Der grundlegende Prozess ist der Aufruf des Client zur Ausführung einer HTTP-Methode, die der entsprechende Server, der zur URI gehört, ausführt, woraufhin der Server das Ergebnis der Anforderung an den Client zurück gibt. Ein prototypisches Beispiel ist die Eingabe einer Adresse wie google.com in den Browser und das Drücken der Eingabetaste. In diesem Fall ist der Browser der Client, der eine GET-Anfrage an die 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 sonst Auswirkungen.
* HEAD ist identisch zu GET, nur dass kein Nachrichtenrumpf gesendet wird. Wenn GET beispielsweise 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 anderen Methoden, die nun folgen, sind optional.
* POST unterscheidet sich von einer GET-Anfrage, insofern dass es einen Anforderungsrumpf hat, in dem Nachrichten bereitgestellt und vom Server angenommen und dann zur Anforderungsausführung vor Rücksendung zum Client verwendet wird. Ein Beispiel dafür ist das Posten einer Nachricht auf einem Forum. Dabei ist der Text der Nachricht im Anforderungsrumpf enthalten.

API

* + PUT hängt in gewisser Weise mit POST zusammen, da es auch Inhalt im Anforderungsrumpf übermittelt. Es besteht jedoch ein semantischer Unterschied, da der Server den Nachrichtenrumpf speichern muss, d.h. der Benutzer 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.

Grafische Benutzeroberflächen

Schließlich können auch grafische Benutzeroberflächen (GUIs) als eine Art API dienen. Die meisten Betriebssysteme haben eine grafische Benutzeroberfläche sowie eine grafischen Darstellung der Ordnerstruktur, einen Desktop, einen Mauscursor und anderen Einrichtungen. In der Anfangszeit der Informatik hatte man nur eine Befehlszeilenschnittstelle und ganz am Anfang nicht einmal die. Heute sind grafische Benutzeroberflächen für die meisten kommerziellen Anwendungen auf einem Desktop oder Smartphone Standard. Eine besonders relevante Anwendungsklasse von GUIs im Rahmen der datenintensiven Wissenschaften sind die Integrierten Entwicklungsumgebungen (IDEs), durch die Programmierer ein gesamtes Supportsystem für ihre Arbeit zur Verfügung haben. Tools, wie PyCharm oder Rodeo, werden in der Python-Programmierung oder von Datenwissenschaftlern, deren Produktivitätsansprüche höher sind, gerne verwendet Diese IDEs bieten Tools zur Code-Vervollständigung, Navigation komplexer Projekte, Quellcodesuche, Codedebugging oder Erstellung eines Performanzprofils des Programms während der Runtime. In gewisser Hinsicht liefern IDEs APIs für alles, was direkt mit den Programmierungsaufgaben zusammenhängt, und die grafische Benutzeroberfläche dieser mächtigen Tools sind unerlässlich.

### Designprinzipien für APIs

Wie wir oben gesehen haben, gibt es APIs also in vielen Formen und Ausführungen. Nun gehen wir in die Praxis über und erstellen eine API. Zudem sollten wir ein Gefühl für gutes und schlechtes Design entwickeln, in dem wir die wichtigsten Designprinzipien für APIs besprechen. Mit einfachen Worten gesagt will jemand, der schreibt, verstanden werden, und vor allem jemand, der Code schreibt. Betrachten wir zunächst zwei konkrete Beispiele eines Machine Learning-Softwareworkflows. Der erste ist ein Beispiel mit Ray, ein viel versprechendes 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, und zur Prüfung, ob die Installation funktioniert hat, könnte 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:

HTTPS

Eine Erweiterung von HTTP, die eine kryptographische, protokollähnliche Transport Layer Security (TLS) bzw. den Vorgänger davon, Secure Sockets Layer (SSL) verwendet, um die Verbindung zwischen den Clients und den Servern sichert, um korruptem Verhalten auf dem Internet vorzubeugen

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 lange ist, um es hier aufzuführen, unterbrochen. Nach der mühevollen Suche in ca. 80 Zeilen Code finden wir schließlich einen Hinweis:

ImportError: Could not import tensorflow

Der Grund ist also, dass dieses Beispiel in Ray die Bibliothek von TensorFlow, herausgegeben von Google, braucht. Die sollten wir also installieren. Das Problem bei diesem Beispiel ist nicht, dass Ray TensorFlow nicht installiert hat, was ohnehin falsch wäre, sondern dass wir unsere Installation verifiziert haben und Ray hat keine Warnung ausgegeben. Unser Setup wurde von Ray problemlos akzeptiert und die Ausführung des Beispiels begann. Zudem ist die Fehlermeldung schwer zu lesen, und Anfänger können solch langen Mitteilungen nur schwer Informationen 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, deshalb kommt die entsprechende Fehlermeldung zu TensorFlow erst nach etwa 50 Zeilen im Programmoutput. Und das ist ein Beispiel für schlechtes Design. Angenommen man startet ein kompliziertes Machine Learning-Experiment mit Hunderten von Codezeilen an einem Abend. Am nächsten Morgen sieht man, dass das Programm sofort abgestürzt ist, weil man eigentlich zuvor TensorFlow hätte installieren müssen. Dadurch kann es zu enormem Produktivitätsabfall kommen, nur weil bei der Fehlerbehandlung eine fragwürdige Designentscheidung 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 eingesetzt wird. Wie bei Ray muss man noch nichts über Machine Learning wissen, denn wir betrachten in diesem Abschnitt nur das API-Design. Nach der Installierung von Keras durch die Eingabe von pip install keras, kann diese durch die Eingabe von

import keras

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

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

Keras deutet also sofort an, dass etwas nicht funktioniert hat. Aber es zeigt auch genau, was notwendig ist, und schlägt in Klartext eine konkrete Behebungsaktion vor, ohne dass man dabei eine Riesenmenge an Textoutput durchsuchen muss. Diese Fehlerbehandlungmethode ist ausgezeichnetes API-Design und zeigt uns, dass gut geschriebene APIs vielen Problemen vorbeugen kann. Eine gute API erleichtert die Programmierarbeit und fördert die Produktivität. Keras wird seit Einführung 2015 für das API-Design gelobt. In den letzten Jahren gilt es nicht mehr so sehr als Bibliothek und viel mehr als API-Spezifikation, die man implementieren und befolgen kann (Sadrach, 2020).

API

2020). Seit dem Release von TensorFlow 2.0, früher eine eigenständige Bibliothek, ist Keras eine offiziell unterstützte API von TensorFlow (Chollet, 2017). Wir kommen später noch einmal auf Keras zurück, um uns weitere gute Designentscheidungen anzusehen.

###### Zen of Python

Einer der Gründe, warum die Programmiersprache Python als elegant und gut für den Anfang gilt, ist die Designphilosophie, mit der es von Anfang an erstellt wurde und die im so genannten „Zen of Python“ verewigt wurde. Zen of Python ist eine Liste flexibler Prinzipien, die man nicht mit einem Regelwerk verwechseln sollte. Sie kann in jeder Python-Version durch Eingabe von import this in der Python-Sitzung abgerufen werden. Die Prinzipien werden im Detail besprochen, da sie viel Aufschluss über gutes API-Design und die dabei auftretenden Schwierigkeiten geben. Nebenbei bemerkt hat sich Daniel Greenfield ein bisschen über diese Philosophie lustig gemacht, indem er die Anti-Zen-Bibliothek mit Namen „that“ ins Leben rief, in der Anti-Pattern, die man vermeiden sollte, aufgeführt werden. Zusammengefasst besagt Zen of Python folgendes:

* + 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 hat Vorteile für die Benutzer.
  + Explizit ist besser als implizit. Wenn dem Benutzer Informationen verborgen bleiben, müssen sie möglicherweise mehr über die impliziten Entwicklerregeln wissen, das heißt es ist besser, so explizit wie mögliche zu sein.
  + Einfach ist besser als komplex, komplex ist besser also kompliziert. Das Sparsamkeitsprinzip von Python, vergleichbar mit Ockhams Rasiermesser, besagt, dass Komplexität wann immer 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 auch nicht zu voll gepackt werden, d.h. Informationsdichte ist zu vermeiden.
  + Lesbarkeit zählt. Code, den man leichter nachvollziehen kann, liest sich natürlich schöner und kann auch leichter langfristig gepflegt werden.
  + Sonderfälle sind nicht besonders genug, um gegen die Regeln zu verstoßen.  
    Dennoch geht Durchführbarkeit vor Regeltreue. Auf Metaebene zeigt uns diese äußerst interessante Zen-Prinzip , dass die Prinzipien nicht ohne Vorbehalte verwendet werden können. Es stimmt natürlich in der Regel, dass Sonderfälle das Design insgesamt nicht ruinieren sollten. Manchmal sind es aber genau diese Ausnahmen zum Regelfall, die unbedingt notwendig sind.
  + Fehler sollten nie stillschweigend übergangen werden, außer sie werden ausdrücklich ausgeblendet. Wenn Fehler schnell passieren, muss der Benutzer nie raten, warum das Programm nicht funktioniert. Dadurch wird das Programmfluss vielleicht 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 man der Versuchung widerstehen zu raten. Anders gesagt sollte man Mehrdeutigkeit vermeiden und dafür so explizit und klar wie möglich sein.
  + Es sollte eine, und vorzugsweise nur eine, offensichtliche Aktionsmöglichkeit geben, obwohl die vielleicht nicht offensichtlich ist, außer man ist Niederländer: Wir nehmen hier Bezug auf den niederländischen Schöpfer von Python, Guido Van Rossum, und das wahrscheinlich kontroverseste Prinzip auf der Liste. Python-Programmierer nennen das „pythonisch.“ Oft gibt es keinen Zweifel an einer pythonischen Lösung und die Community stimmt bei den meisten Mustern überein. Aber manchmal ist es nicht klar, ob ein Ansatz tatsächlich besser ist als ein anderer.

Dennoch sollte es das Ziel sein, 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 dagegen gleich eine Lösung finden, selbst wenn diese nicht perfekt ist. Sie kann später durch Iteration verbessert werden. In Notfall sollte man jedoch nicht einen Weg einschlagen, 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. Nicht alles, was sich leicht erklären lässt, ist gut, aber wenn man es nicht erklären kann, ist es auf jeden Fall schlecht.
    - Namespaces sind eine tolle Idee-davon brauchen wir viele! Python setzt sehr häufig Namespaces zur Modularisierung und Struktur des Codes ein. Möchte man zum Beispiel einen Zoo mit vielen Tieren in Python implementieren, könnte die Klasse Elephant mit dem Befehl from zoo.animals.mammals import Elephant statt durch from zoo import Elephant importiert werden, da durch letzteres vielleicht zu viele Begriffe im Namespace eingeführt würden.

###### Trennung von Schnittstelle und Implementierungen

Trennung von Zuständigkeiten

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

Ein allgemein gültiges Prinzip beim Schnittstellendesign ist die Trennung von Schnittstellen und ihren Implementierungen. Streng genommen ist eine API den besprochenen Definitionen und Eigenschaften zufolge einfach eine Schnittstelle, und der Zweck einer API ist es, dass Benutzer sich nicht um die Implementierungsdetails kümmern müssen. In der Praxis sollte man sich jedoch immer wieder daran erinnern. Die Trennung der Schnittstelle kann weitgehend im Rahmen des Prinzip der Trennung von Zuständigkeiten gesehen werden. Stark typisierte Sprachen wie Java oder Scala haben dedizierte Schnittstellenklassen, durch die diese Praktik unterstützt wird. In Java kann eine Schnittstelle keine konkrete Implementierung habe, da dies gezielt nicht erlaubt ist. 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 kompatiblen Bytecode auf der Java Virtual Machine (JVM) kompiliert werden. Java-Klassen im besonderen können in den Scala-Code importiert werden. Darauf kann Software 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. Man muss in Python trotzdem die Objekte, mit denen Benutzer interagieren, die öffentliche API des Codes, den restlichen Code und die Implementierung gut durchdenken. Wenn man die öffentliche API als Oberfläche betrachtet, auf der die meisten Benutzer arbeiten, ist sie für größere Projekte nur die Spitze des Eisbergs und gutes Schnittstellendesign wird umso wichtiger.

API

###### 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 Benutzer die Interaktion erleichtern. Um es mit dem bekannten Computerwissenschaftler Edsger Dijkstra zu sagen „abstrakt ist etwas ganz anderes als vage... Der Zweck von Abstraktion ist nicht vage zu sein, sondern eine neue semantische Ebene zu schaffen, auf der man absolut präzise sein kann“ (Misa & Phillip, 2010, S.1). 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 gemacht ist, 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. Selbst wenn man noch nie etwas von Deep Neural Networks gehört hat, könnte man mit der obigen Information folgende Keras-Modellspezifikation verstehen:

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, fügt man sequenziell so viele Ebenen hinzu, wie man möchte. Die Details des Modells sind hier nicht relevant, man sollte jedoch bemerken, dass der Code sich fast wie Klartext liest und genau wie die gleiche, eben beschriebene, hohe Abstraktionsebene, d.h. Modelle und Ebenen, funktioniert. Man könnte meinen, dass das offensichtlich ist. Das ist es jedoch nicht: Vor Keras hatte keine der Deep Learning-Frameworks einen ähnlich simplen und intuitiven Programmierstil, wodurch der Fachbereich des Deep Learning für Fachleute bei weitem nicht so zugänglich war. 2020 hat sich die Situation verbessert, aber die Auswirkung von gut geschriebenen APIs, wie Keras, auf den Wirbel um Deep Learning und Machine Learning im Allgemeinen darf nicht unterschätzt werden. Bekanntermaßen sind alle Abstraktionen undicht (Spolsky, 2002), was heißen soll, dass, ganz gleich wie gut eine Abstraktion in der Informatik durchdacht ist, es immer Fälle geben wird, bei denen man genauer nachsehen muss oder eine Ausnahme machen muss. Dadurch wird die Unordnung in der Welt widergespiegelt. Abstraktionen sind nur ein Versuch, Dinge in Kategorien zu gruppieren, und das ist ein Prozess, der aus verschiedenen Gründen schief gehen kann. Diese Aussage ist zwar wahr, sie spielt in der Praxis aber oft keine Rolle: Mit guten Abstraktionen kann man viel erreichen, während schlechte Entscheidungen oft zu miserablen Ergebnissen führen. Auch folgendermaßen könnte man argumentieren, vor allem, wenn es um undichte Abstraktionen geht: man braucht eben verschiedene Abstraktionsebenen. Die höchste Abstraktionsebene ist oft gut genug, aber manchmal muss man ein paar Ebenen tiefer gehen, um den Code an den Anwendungsfall anzupassen.

###### Fehlerbehandlung

Am Beispiel der Ray und Keras-Python-Bibliotheken haben wir bereits gute und schlechte Fehlerbehandlung gesehen. Aber eine Generalisierung und Zusammenfassung einiger wichtiger Punkte lohnt sich:

* + - Wenn Fehler auftauchen, ist es am besten, wenn das so früh und schnell passiert, wie möglich. Dadurch wird maximale Transparenz erreicht und ernste Fehler später im Programm, die oft schwieriger zu analysieren und beheben sind, vermieden.
    - Fehlermeldungen sollten menschenlesbar und klar geschrieben sein. Sie sollten dem Benutzer 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 evtl. nicht mehr vorhanden sind, geben.

###### Konvention und Konfiguration

Laut Computerwissenschaftler Phil Karlton „gibt es nur zwei Dinge, die in der Informatik schwer sind: Cacheinvalidierung und die Benennung von Dingen“ (Fowler, 2009, S. 1). Obwohl das nicht unbedingt ganz ernst gemeint war, ist diese Aussage nicht unbedingt falsch. Auf das API-Design trifft hauptsächlich ein Teil des Zitats zu, nämlich dass Dinge zu benennen schwer ist. Gute Namenskonventionen zu finden ist überraschend schwer, und einige Sprachen, wie Java, haben die Tendenz in sich sehr lange Klassennamen zu haben. Als Beispiel soll hier AbstractTransactionalDataSource- SpringContextTests aus dem Spring-Framework angeführt werden. Das ist ein Beispiel, das man vermeiden sollte, und deutet entweder daraufhin, dass zu wenig begriffliche Klarheit in der Abstraktion bestand oder dass die Klasse mit zu viel belegt ist (und damit auch das Prinzip der Trennung der Zuständigkeiten verletzt wird). Klare und präzise Namensgebung der Klassen und Funktionen sollte beim Softwaredesign stets berücksichtigt werden. Neben den Namenskonventionen spielen auch andere Konventionen eine Rolle. In Python kann man beispielsweise auch Standardargumente in die Funktionssignatur schreiben. Davon sollte man immer dann Gebrauch machen, wenn sinnvolle Standards vorhanden sind. Die meisten Benutzer sind keine Poweruser und interessieren sich nicht für einzelne Parameter, die angepasst werden können. Es ist jedoch besser, in den Funktionsparametern viele konfigurierbare Parameter zu haben, als sie Benutzer durch Code modifizieren zu lassen. Hier ist ein konkretes Python-Beispiel. Es gibt eine einfache Klasse mit einem Bucket, das eine Größe hat und mit etwas gefüllt werden kann. So würde die Python-Klasse dafür dargestellt:

class Bucket:

def init (self, size=10): self.size = size

def update\_size(self, size): self.size = size

API

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

bucket = Bucket()

Sie können auch die Größe des Buckets konfigurieren:

bucket = Bucket(size=20)

Und schließlich könnten sie erst ein Bucket erstellen, dessen Größe sie durch folgenden Code anpassen können (allerdings ist dies ein fragwürdiger Vorgang)

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 abgedeckten Prinzipien laufen auf logisches Denken und gesunden Menschenverstand angewendet im Fachgebiet Informatik hinaus. Denn bei guten APIs oder bei gutem Design im Allgemeinen geht es immer darum, sich in den Benutzer hinein zu versetzen und die kognitiven Faktoren zu berücksichtigen, die bei der Erstellung von Produkten für Menschen eine Rolle spielen. Mit anderen Worten, das Benutzererlebnis - auch User Experience oder, wenn es um die Beziehung zwischen Ingenieur und Code geht, auch speziell Entwicklererlebnis - ist wichtig. Das Cognitive Dimensions Framework (Clarke, 2004) führt insgesamt zwölf Punkte auf, die beschreiben, wie Informatiker 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 hauptsächlich arbeiten?
  + Lernstil: Welche Anforderungen bestehen, wenn man die API erlernen möchte? Ist die Lernkurve steil? Was ist die Entwicklerzielgruppe und welches Vorwissen haben 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 vom Benutzer 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 alleine erlernen? Ist sie gut dokumentiert und kann man die Komponenten intuitiv nachvollziehen und zusammenbauen?
  + API-Ausführung: Wie gut passt die API auf die jeweiligen Bedürfnisse? Kann sie flexibel konfiguriert werden oder muss sie stark an die Bedürfnisse des Benutzers angepasst werden?
    - Schwerfälligkeit der API: Ist es leicht, die API an den Anwendungsfall anzupassen und zu erweitern? Wird die Sprache der angestrebten Entwicklercommunity verwendet?
    - Konsequenter Aufbau: Kann man aus einem Teil der API Schlüsse auf die Funktionalität eines verwandten API-Teils ziehen? Wurden also 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 Benutzern dringend erforderlich ist. Folgende Aspekte sollten stets abgedeckt werden:

* + - Dokumentation des Codes
    - Dokumentation der Schnittstelle, Eingabeparameter und Rückgabewerte
    - Dokumentation der gedachten Verwendung der API

### 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 sogar relativ kompliziert ausfallen, so dass wir zeigen können, wie gutes Design einer API diese Komplexität verdecken kann. Hier ist unser konkretes Beispiel: Sie führen eine Firma, 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 Ihren Benutzern diesen Algorithmus anbieten, um deren Bedarf zu decken und damit Geld zu verdienen. Der erste Schritt ist die Erstellung einer Python-Bibliothek, in der der Algorithmus sauber gespeichert werden kann und anhand der er für Benutzer zugänglich wird, ohne dass der Algorithmus selbst, der vertrauliches Wissen darstellt, 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 Google 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:

API

1. Einen Stapel Dokumente nehmen und die wichtigen Elemente (z.B. die Wörter im Dokument) transformieren oder zuordnen (von Englisch „map“) nach einer vom Benutzer 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 aufgetreten. 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 sollen, 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 dann reduziert („reduce“) wird. Aber der zweite Schritt ist ebenfalls wichtig. Die drei Schritte mögen einfach aussehen, das Potenzial liegt jedoch darin, dass sie über Hunderte von Maschinen massiv parallel ausgeführt werden können. Als nächstes erstellen wir eine simple Implementierung dieses Algorithmus für unseren Anwendungsfall, d.h. Wortzählung. Wir beginnen mit der Implementierung der drei Phasen. Als erstes definieren wir die Zuordnungsfunktion, die (word, 1) als Schlüssel-Wert-Paar für jedes Wort im Text zurück gibt:

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:

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:

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:

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:

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:

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, erhalte wir das folgende Output:

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 gedacht 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 Benutzern 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 die Benutzer genau 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, die wir den Benutzer liefern können, beginnen:

API

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 nur Zugriff auf count\_words\_naive, nicht jedoch auf den implementierten Code. Aus der ersten Interviewrunde mit potenziellen Kunden haben wir folgendes Feedback, das zur Ansatzverfeinerung verwendet werden kann:

* + Viele Benutzer 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 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, 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 kannten Python nicht gut genug und wollten einfach eine Dateiliste statt den einzelnen Dokumenten in Python laden.

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

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:

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 hatten, 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 separaten Codebasis zu erstellen, ist auch eine gute Designpraktik.

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 conta 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 gib 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.

API

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 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 Lerneinheit wurden Grundlagen, Design und Implementierung von Anwendungsprogrammierschnittstellen (APIs) unter Berücksichtigung datenintensiver Projekte behandelt. Alle Programmierungsbeispiele in dieser Lerneinheit waren in Python, da es leicht ist, einen Prototyp in Python zu erstellen und da Python derzeit die relevanteste Sprache für Projekte in Data Science und verwandten Gebieten ist. Im ersten Abschnitt gingen wir auf die grundlegende Terminologie für APIs ein und auch darauf, welche Rolle verschiedene Abstraktionsebenen, vom Betriebssystem und Hardwarezugriff auf moderne Web-Schnittstellen, bei der Arbeit mit Computern spielen. Diese API-Ebenen decken eine breite Klasse an wichtigen Einstiegspunkten für die Anwendungsentwicklung ab. Dazu gehören Datenbankzugriff, Programmiersprachen und die Verwendung der bevorzugten Toolkette in datenintensiven Fachgebieten. Von jetzt an betrachten wir APIs als Spezifikationen zwischen einem API-Designer und dem Benutzer, also als einen Vertrag, an den die Parteien gebunden sind. Dieser Abschnitt konzentrierte sich auf die Verwendung von Bibliotheken und Frameworks und die wichtigsten Bausteine von RESTful Webanwendungen. Danach haben wir uns angesehen, was zur API-Entwicklung gehört, indem wir die wichtigsten Designprinzipien sowie gute und schlechte Praktiken beleuchtet haben. Auch wichtige Elemente, wie die Trennung der Zuständigkeiten, 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 darauf folgenden Abschnitt befassten wir uns mit einem Anwendungsfall einer kleinen Bibliothek in Python, die Benutzern zur Verfügung gestellt werden sollte. Ein relativ komplexer, bereits bestehender Algorithmus, mit dem Wörter in Dokumenten gezählt werden können, wurde in eine neue, für den Benutzer sehr kleine API eingebaut. Es wurde weiterhin gezeigt, wie für den Benutzer uninteressante Funktionalität effektiv versteckt, Code klar dokumentiert, Typüberprüfung von Eingabe- und Ausgabeargumenten verwendet und Fehlerhandhabung zur Aufdeckung von ungewolltem Verhalten eingesetzt werden. Sie können also nun bei der Arbeit mit APIs deren Stärken und Schwächen beurteilen, APIs effektiv erstellen und eigene Bibliotheken und REST-Schnittstellen in Python einrichten.



# Lerneinheit 5

## Vom Modell zur Produktion

#### LERNZIELE

Nach Abschluss dieser Lerneinheit verstehen Sie ...

… the details of the process of developing a machine learning model, and how it differs from the standard process of software development.

… the lifecycle of a model once it reaches production, including updating, monitoring, health checks, performance, and scaling.

… the speciﬁcs of deploying a model and keeping it deployed.

… the purpose of MLOps and DataOps.

… how end-to-end cloud services offer complete solutions to all the issues above, speciﬁcally covering AWS SageMaker.

DL-E-DLMDSSEDIS01-U05

1. Vom Modell zur Produktion

### Einführung

A machine learning solution provides predictions based on a set of input data in a non-deterministic way. Its development is an iterative and explorative process. We begin by identifying and clarifying a case where machine learning can bring value to an organization. We then proceed to collect and explore the available data, generate a hypothesis on the nature of the data, the problem, and the solution, that we then explore and test both visually and programmatically in machine learning models. Once we have a solution candidate, we proceed to integrate it in the existing infrastructure, or develop software to take advantage of it. We will explore how the machine learning development process has similarities and differences with the traditional software development process. Bringing a machine learning solution to production presents speciﬁc challenges due to its non-deterministic nature, the nature of the teams devel- oping it, and its intrinsic dependence on the data used to train it, as well as the data it is used on. We will explore the speciﬁc challenges of version control, testing, and moni- toring due to the nature of the machine learning solution. Later, we will begin to see how some tools offer support for several of the challenges of production machine learning, introducing the model of MLOps, and how continuous integration (CI) and continuous delivery (CD) models of DevOps can be applied and extended to machine learning solutions, with the addition of continuous training (CT). We will speak in more depth on the existing solutions with MLFlow, Kubeﬂow, and Michelangelo. We will then explore how existing cloud providers offer solutions to the challenges we are facing, going into more speciﬁc detail in the case of Amazon SageMaker.

### Model Development Lifecycle

Machine learning

model This is a ﬁle that has been trained to rec- ognize certain types

of patterns.

In this unit we are concentrating on the challenges of bringing a model to production, so it is useful to start from an overview of the lifecycle of model development. Machine learning involves creating a model trained on certain training data which can afterward process additional data to make predictions. In this context, a machine learning model, or a statistical model, can be considered a black box that, given a set of inputs, returns some outputs. The basic idea is that the model allows us to map a set of inputs to a desired output. For example, we want to train a computer to classify an image as a cat or a dog. In this example, we give the model input images and labels telling the model whether the image is a cat or a dog. The black box then learns an association between inputs and labels, so that it can hopefully predict the right answer (cat or dog) for pre- viously unseen images.

To develop a model that brings value to an organization, the basic steps are

1. understanding the problem,
2. ﬁnding good assumptions and hypotheses,
3. collecting the available data,
4. exploring it to see if the assumptions and hypothesis seem realistic,

Vom Modell zur Produktion

1. experimenting with possible models and data processing steps to solve our prob- lem,
2. training a model, and ﬁnally
3. deploying it on a development pipeline to make sure it behaves as expected.

Let’s delve deeper, and please note that all steps are connected to each other and can’t be seen in isolation. As an example, while cleaning the data, we also analyze them a bit, and after exploring, we sometimes need to go back and change our cleaning proc- ess, because it removed too much information together with the noise, or because it didn’t remove enough.

###### Problem Understanding and Hypotheses

First, it is essential to understand the problem you’re dealing with. In the best of cases we are given a very narrowly deﬁned project scope. However, requirements are often vague, like “improving turnover,” “reducing churn,” or “identifying customers that won’t pay.” Finding the right answers to what is actually needed to succeed with your project is called requirement engineering. Various project management techniques need to be carefully applied to achieve this. The process of understanding and deﬁning the prob- lem can involve long interviews with matter experts, design thinking, and exploratory modeling. We often have to go back and forth checking and correcting our initial hypotheses and assumptions. The ﬁeld of data science is still comparatively new and many companies struggle to adopt a data-driven mindset, meaning their stakeholders have a hard time setting up data science projects for success. For this reason, it is important to have a clear idea of what we want to solve and have clear performance indicators. This can be done by establishing well-deﬁned, speciﬁc problems that can be answered. Rather than simply “identifying problematic customers,” a goal such as “we want to reduce the number of customers not paying after 2 weeks by at least 20 per- cent” is much more concrete. As another example, even something that sounds more speciﬁc, such as “identifying the vehicles entering a building” could be improved upon to set the goal of “at least 95% of the vehicles entering a building are correctly identi- ﬁed.” Some problems can be very hard to deﬁne, as in the case of machine translation: How do we score translations from best to worst? The answer to that is highly subjec- tive, and while we somehow know, as humans, when a translation is really bad, it is difﬁcult to evaluate good ones. Should they be very close to the original, should they keep the essence and take liberties, or something in the middle? Once we have deﬁned the problem, we can start making hypotheses about what could solve it, what we could need to accomplish it, what assumptions we need to check, what correlations to have for mapping inputs to outputs, and begin researching different possibilities to accom- plish that.

###### Data Collection, Cleaning, and Wrangling

Once we understand the problem and have a hypothesis for how to tackle it, we have to collect the data to work with. In some cases, collecting data can be as simple as getting access to a database with all the data tagged in the way we need. In other

ETL (extract, trans-

form, load) This is a type of data integration that refers to the three steps used to blend data from multiple

sources.

Data scraping This is a technique in which a computer program extracts data from human- readable output coming from another

program.

Exploratory data analysis (EDA)

This is an approach to analyzing data sets to summarize their main characte- ristics, often with visual methods.

cases, it is much more complicated and can often involve a process of ETL (extract, transform, load). In some other cases, we can realize the organization does not have the data we need, but they can be accessed from a public source. There are several open data repositories, with more being added every day. The problems with open data are that, since they are open, they generally won’t offer much of a competitive advan- tage and, above all, they rarely meet our exact needs.

Another possibility is collecting the data ourselves. This is generally more time and resource intensive, but it allows us to obtain exactly what we need to solve the prob- lem, or at least the best we can get. Examples include

* collecting movement data from passengers,
* taking photos from the camera we plan to ship with a product,
* equipping sensors on devices to monitor their behavior, or
* getting humans to tag pieces of existing data for supervised learning.

How to properly collect the data we need is both an art and science in itself, often overlapping with experiment planning. Another, often simpler possibility is to do data scraping, i.e., retrieving data from pages on the web or from sources not aimed at being directly consumed by computers, and transforming the data into computer-useable for- mats. It can involve getting data from websites, PDFs, or scanned documents. Scraping is time intensive, since it requires the creation of speciﬁc software for each individual source, and brittle since it is very tightly coupled to the source. Even a small change in the source will break the process and require it to be adapted. Additionally, scraped data can present legal problems that need to be evaluated if the data was not origi- nally collected for the intended purpose (Import.io, 2017). Any sort of data (scraped, collected, or accessed from a database or datastore) will need cleaning and re-forma- tting for the intended purpose. This is another extensive topic, known as data wrang- ling or data munging. Data wrangling, or munging, is the process of transforming and mapping data from one data format into another to make the data more appropriate and valuable for our purpose. It involves transformations typically applied to distinct entities (ﬁelds, rows, columns, data values, etc.) within a data set, and could include actions such as extractions, parsing, joining, standardizing, augmenting, cleansing, con- solidating, and ﬁltering to create desired outputs (Wikipedia, 2020c).

Exploratory data analysis

Once we have access to data, we can start working with them to get an idea of what they can tell us. This step is generally called exploratory data analysis (EDA), and it refers to the process of performing initial investigations on data to discover patterns, spot anomalies, test hypothesis, and check assumptions, using summary statistics and graphical representations.

Standard tools for EDA help us to summarize the type and number of (distinct) values, type of data points, and median and mean of the data; create visualizations for an intuitive idea of what the data look like; and see what patterns can be extracted. Since humans are very visual, it is much easier for us to see patterns on an image than in thousands, or millions, of lines of data. EDA is an exploratory process; a data scientist will explore different possibilities, and delve deeper when something looks interesting.

Vom Modell zur Produktion

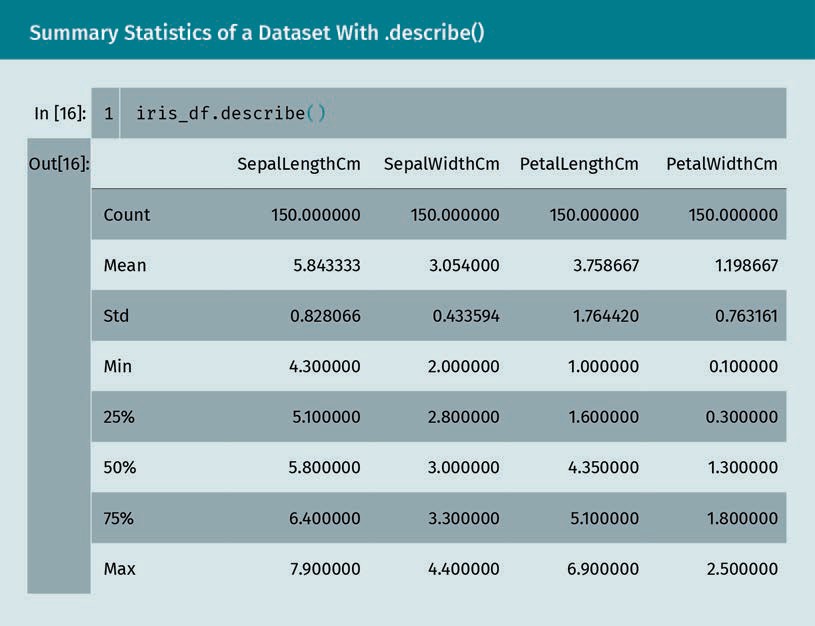
It is an intuitive process that requires experience, so we can only give a simple example of it. We will perform a bit of exploration of the iris dataset, a small dataset containing three classes of 50 instances each, where each class refers to a type of iris plant and is often used for demonstration purposes (Fisher, 1936). Typical ﬁrst steps in EDA are che- cking the look of the data by inspecting the ﬁrst few rows, which we do with pandas in Python (Pandas, n.d.). Note that we downloaded the iris dataset ﬁrst in CSV format and stored it locally as a ﬁle called iris.data.csv. This is what the data look like in tabu- lar form:

import pandas as pd

iris\_df = pd.read\_csv('iris.data.csv') iris\_df.head() # first 5 rows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| First Lines of a Dataset with .head() | | | | | |
| Out[15]: | | | | | |
|  | Sepal- LengthCm | Sepal- WidthCm | Petal- LengthCm | Petal- WidthCm | Species |
| 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 |

We see the dataset has four numeric values, and one text label called “Species.” We can compute summary statistics for the numeric values as follows:



We can conﬁrm we have a total of 150 values and get an idea of the average and distri- bution of values.

###### Experimentation and Feature and Model Selection

Feature selection This consists of selecting a subset of relevant features to be used in the

model.

Once we have the data loaded in a useable way and we have an idea of what the data can tell us, we can start experimenting with models. We will test several different mod- els, starting from simple models like linear or logistic regression, to more complex, like neural networks and XGBoost, depending on the use case. In the experimentation cycle, we will start with a few models and initially test them to get an idea of how they per- form. Part of the process is selecting possible features. The data we have can contain meaningless noise, and some information could be extracted given our knowledge of the target domain. For example, if our data are about trafﬁc, the day of the week is an important factor, but if our dataset contains only the day of the month, we will have to calculate the day and add it to the features. On the other hand, training a model incurs the curse of dimensionality. The more dimensions we have, the harder it is for a model to converge, so we sometimes want to perform dimensionality reduction. At this point we also usually begin considering possible models. The main type of machine learning problems are supervised, unsupervised, and reinforcement learning.

Vom Modell zur Produktion

* + Supervised learning maps input data to known output data. Example: categorizing images as cats and dogs, or predicting the stock market.
  + Unsupervised learning explores patterns in your data. Example: clustering docu- ments by topics by looking at similarities in those documents, without specifying the topics beforehand.
  + Reinforcement learning studies how agents interact with their environment by rewarding favorable situations and punishing bad ones. Example: learning to play a video or board game.

Choosing the right machine learning model for our task isn’t easy. There are many machine learning models, and several Python software packages offer access to some, or many, of them. We tend to differentiate between traditional machine learning algo- rithms, mostly based on statistical methods, and deep learning algorithms, based on neural networks. In many cases, we get even better results with ensemble models that aggregate the results from several models to improve accuracy. It is sometimes the case that a ﬁrst layer of machine learning is done by deep neural networks, and their results are then aggregated by simpler traditional methods. Once we have selected what model to try, we divide our data into train, test, and validation datasets, often with techniques like k-fold to maximize the amount of data we can use to train and val- idate.

###### Hyperparameter Tuning

Every machine learning model has a set of parameters that are automatically adjusted during the learning process, and which we don’t manually tweak. However, there are other, model-speciﬁc parameters called hyperparameters that can be chosen by the users of an algorithm. For example, in the case of deep neural networks you can specify the depth of a network. The way in which you select your features that will be fed into the model can also be seen as a hyperparameter of your machine learning experiment. Deciding on which hyperparameters to use is not easy, as the training process usually takes a lot of time, so you can’t compute all possible combinations. Luckily, there are algorithms and libraries that help you with hyperparameter tuning. Using such a tool, the result will be hyperparameters optimized for the current dataset and application.

###### Training

We can now start training different models to compare their performance. We will usu- ally train several models on the dataset for different choices of features. If our dataset is very large and training takes a lot of time, it is often useful to use only a subset of the dataset initially. Once you’re conﬁdent in your choices (preprocessing, model, and feature selection), you can train your model again on the full data set. Depending on the amount of data and the use case, this can take quite a while, and lots of resources. To give an extreme example, GPT-3, by OpenAI, has 175 billion parameters and would require 355 years and $4,600,000 to train on current hardware (Brown et al., 2020). Our cases will not be as extreme, but we still have to do a complete training on the models

Dimensionality reduction

This is the transfor- mation of data from a high-dimensional space into a low- dimensional space while still keeping some meaningful properties of the original data.

Ensemble models These use multiple models to obtain better predictive performance than could be obtained from any of the con- stituent models alone.

Train, test, and vali- dation datasets The test dataset is used for the initial ﬁtting of the model, the validation data- set to evaluate the performance of the model at different steps, and the test

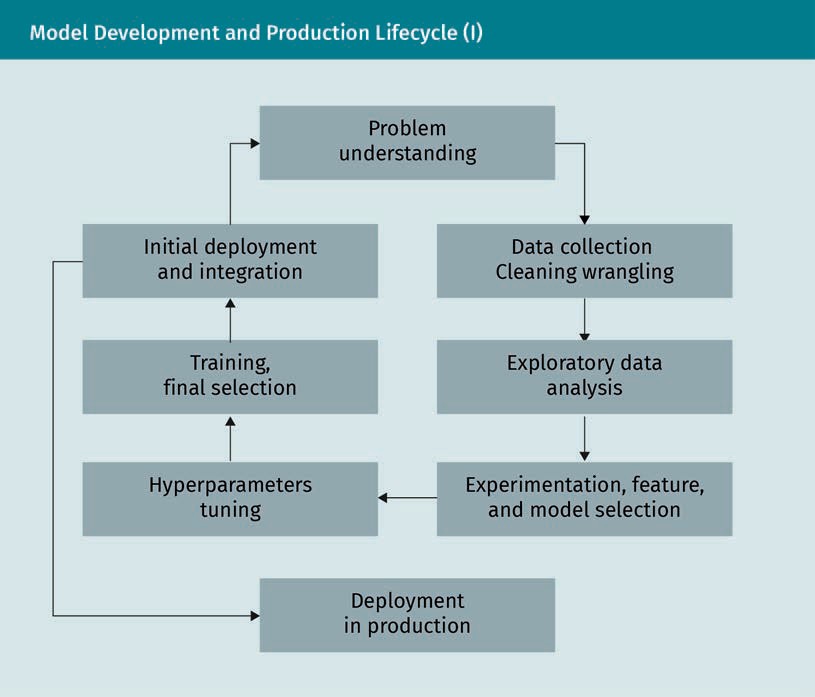
dataset is used for a ﬁnal unbiased evalu- ation of the per- formance.

Hyperparameter This is a parameter whose value is used to control the learn- ing process.

we think are good candidates. This of course does not apply for models that can train in less than a few hours. At the end of the process, we have our trained models, and we can check their performance using the test dataset.

###### Initial Deployment and Integration

Once the model is ready, we need to make it available somehow. The easiest and most common way is to wrap it in a REST interface and deploy it as a microservice in a con- tainer. Once this is done, we can do some basic tests of usability, connect it to other services, and set it up in a development pipeline to make sure it behaves as expected.



###### Differences to Software Engineering

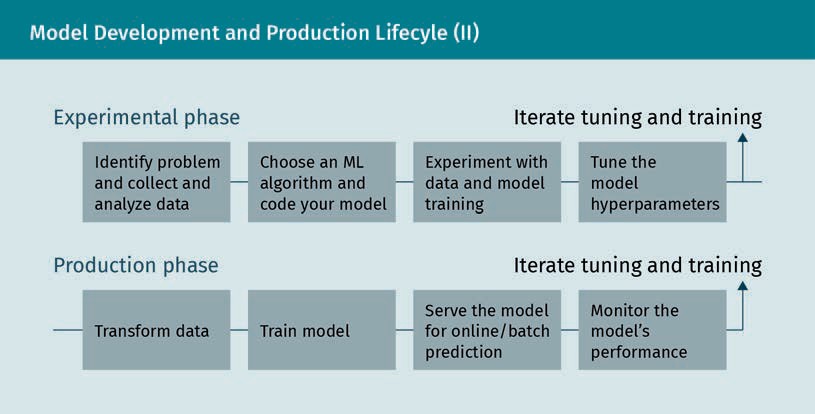
Let’s compare the traditional software engineering lifecycle to the machine learning lifecycle. In both cases it is necessary to understand the problem to be able to solve it correctly. However, in the traditional software world, the problem is often better deﬁned, at least at the level of pure implementation. Engineers use quite common building blocks, and can iterate and check their assumptions. In traditional software

Vom Modell zur Produktion

engineering we talk less about hypotheses and more about requirements. It is easier to implement techniques for requirements-based testing as the results are less “fuzzy.” The code is supposed to be deterministic, unlike machine learning models, even if some interactions can add complexity.

### Model Production Lifecycle

Once our models are ready for production, the production requirements have many similarities to the production lifecycle of traditional software, as well as some speciﬁc differences. As with all code, it should be integrated in a continuous development and deployment pipeline, allowing for quick turnarounds and frequent shipping of reliable, secure, and available software. Using machine learning models to solve problems has been called Software 2.0 (Karpathy, 2017). The complete ML system contains: the code to create the model, the trained model itself and other possible artifacts, and the data used to create the model. This code must be integrated in the pipeline described above. To bring and keep models in production, we then have to take care of versioning the code and storing and deploying all of the code. Additionally, we take care of the endpoints using our models, persist our predictions, log the results, and monitor health, performance, and what happens when our models are used on new or unseen data.



###### Models in Production

When models reach production, in addition to all the usual challenges of production software, they encounter some speciﬁc ones due to the fact that they are based on software. We can think about some of the services most of us use practically every day.

Software 2.0

This uses machine learning, speciﬁcally neural networks, to create a fundamen- tal shift in how soft- ware is written.

Artifacts

An artifact is the output created by the training process.

Online search engines are, at their core, a machine learning system. Given a query, in the form of a text string, its goal is to return links to the web pages that will most likely match the intent of the user. The results have to be accurate, fast, and up to date, and the system needs to be always available. It is constantly updated on new data collected from the web, and it is adapted to the preferences of the user. Some movie streaming websites contain a recommendation system that aggregates what other users watched, combined with the history of a speciﬁc user, to try to propose what the user could want to watch next. Its goal is to keep the user engaged (some could think it performs too well, as many of us can attest after too many hours of watching a series). It needs to be constantly available, serving huge amounts of data, and constantly adapting to the changing preferences of the users. Different challenges are encountered by digital ﬁnancial and payment services. They need to constantly monitor transactions to ﬂag inappropriate and fraudulent ones, with very little delay and high accuracy; rejecting good transactions annoys users, while accepting bad ones is costly. These systems need to have several failsafe mechanisms and be constantly monitored for perform- ance and delays.

###### Versionsverwaltung

Git GitHub is a popular platform based on Git, hosting many open-source

projects.

Version control sys-

tem This keeps track of changes, including what was changed when and by whom, and can compare different versions.

Version control is the management of changes to documents, computer programs, and other collections of information. The reason we need versioning is to track changes, be able to revert to previous working solutions, and be able to coordinate updates. We are already familiar with Git for version control, and many tools aimed at the machine learning process either use it or are based on it. We usually identify the changes via numbers or letters, called revisions or revision numbers. An initial version of a set of ﬁles or code could be “revision 1.” After a change is made, we will have “revision 2,” and so on. Using versions, and a version control system, allows us to keep track of changes (speciﬁcally what was changed when and by whom), compare different versions, and synchronize changes between different parts of an application.

###### Versioning Code

When considering the code purely as a “programming language,” there are two different kinds of code in the case of an ML system: implementation code and modeling code. The implementation code could be glue code, the code used to access APIs, or system integration code which connects the ML system to the other applications. Modeling code is used for model development. In some cases, the code can be written in several different languages. This code needs to be versioned for releases, and the releases tes- ted. Additionally, code has dependencies that need to be updated, and it will interact with other pieces of code. So far, this is exactly what happens in any normal software development process, where the concept of infrastructure as code (IaC) has been developed to solve the problem of environment drift in the release pipeline. However, as we have noticed, the ML system is not only the code to generate the model and the code to wrap it and connect it to other services: It includes the data used to generate the models, and the models themselves, which all need versioning.

Vom Modell zur Produktion

###### Versioning Data

The data used to train the model can change. The metadata (format, names, order, and number of columns) can change, values can be dropped or added, precision for some values can change, or the statistical characteristics compared to the new data can change too much compared to the data used to originally create the model. To be able to generate the same model over and over again, a requirement for reliable deploy- ment, we need to make sure we are using the same data that we used when we ship- ped. This means versioning the data. As noted, the data includes both the data and the metadata. The main issue with the data is size: The data used to train a model can span to a few megabytes, or even several petabytes in some cases. Storing a few mega- bytes in a normal version control system isn’t a problem; however, storing gigabytes (or more) is. This has given rise to several systems to store larger ﬁles, from the open source add-ons to Git (like Git Large File Storage (LFS) (GitHub, n.d.-c) and Data Version Control (DVC) (DVC, n.d.)), to several closed source solutions. The two main issues of versioning the data are storage requirements for potentially huge amount of data, and the task of keeping the metadata matched with the requirements of the model.

###### Versioning Models

The models, and other artifacts generated during the model development, also need versioning, so that we can match them with the algorithms used to generate them, the code that supports them and that uses them, and the data used to train them. The storage requirements of models are not as problematic as the ones for the data, even if models can still require hundreds of megabytes, posing a challenge for normal version control systems. Git has a maximum ﬁle size limit of 100mb, but we have already men- tioned Git LFS as a workaround for that (GitHub, n.d.-c).

###### Reproducible Model Training

It should be possible to rebuild the model on demand starting from a speciﬁc version of the data and the code used to generate the model, using the speciﬁc version we desire. However, since this takes time and resources, we want to store the version of the model together with the rest. Versioning ensures that we can always reproduce the same results at different points in time and on different machines, starting from the same version of data and code.

###### Deployment

Once developed and trained, the model needs to be deployed for it to be of any use to the organization. We have two main possibilities.

Metadata

This tells us about the nature of a cer- tain column (or fea- ture) in our dataset.

* + - Embedded model: The model as an artifact (the actual ﬁle) is built and packaged together with the application using it. From now on, it is considered a part of the application resources, the same way a background image for a website would be.
    - Model deployed as a service: The model is wrapped in a service that can be deployed independently of the consuming applications. This allows for decoupling, but can increase latency, and the applications need to run some sort of remote invocation.

In both cases, the models need to be stored and once built, delivered to the appropri- ate infrastructure. Deploying as a service adds some complexity, but we can think of the service as an application: In such a case, the model artifact is stored and deployed embedded with the serving application. The main considerations for deployments are practically identical to the ones in traditional CI/CD pipelines for software develop- ment, including integration tests and system tests. Still, model deployment can present different complex scenarios.

Multiple models

We can sometimes have more than one model for the same task, or several models working together for the same task. We can choose to deploy the models as separate services, or accessing multiple models with a single API call. The ﬁrst possibility offers more ﬂexibility, but the consuming applications then need to be extended, while the second scenario allows us to change the number and type of models without rewriting the rest of the application.

Shadow models

In some critical scenarios, it can be desirable to have several versions of the same model in production side by side. This allows us to make sure that the new models, that we expect to perform better than the old ones, actually do. To ascertain it, we gather data on the shadow model behavior before moving it to active production, a procedure similar to testing, but on live data: This way, we can make sure the behavior of the new models is as expected before going live.

A/B test This test consists of

a randomized experiment with two variants, A and B.

Competing models

In a more complex scenario, we can keep multiple versions of the same model in pro- duction to ﬁnd out which one is best, like an A/B test. This adds complexity in the form of infrastructure and routing to make sure that the trafﬁc reaches the right models, and to make sure we have enough data to make decisions on what model to keep.

Online learning models

The models we discussed so far are developed, trained, and deployed. Online models get updated constantly and can continuously improve their performance with new data, learning while in production. This presents extra complexity: the models aren’t static artifacts, they won’t yield the same results. We need to make sure the model per- formance doesn’t degrade, that the data used for training is acceptable, and we need versioning of updated models and production data to be able to revert to a version with good performance in case something goes wrong. It is a quite advanced scenario, rarely used outside of very specialized applications, usually in very big organizations.

Vom Modell zur Produktion

To support more complex deployment scenarios, it is advisable to use elastic infra- structure that can scale on demand. This involves using cloud providers which will be covered in a later section.

###### Testing and Quality

In software engineering, testing allows us to increase our conﬁdence that our system behaves correctly, and allows us to be warned when something breaks. Testing in an ML environment has a bigger scope than in traditional software engineering, since it covers data and training, in addition to the integration with other components.

* + Data: We can add tests to validate input data against the expected schema, or to validate our assumptions about valid values. We want to be sure the data being fed to the model corresponds to the kind of data the model was trained for. This is even more important if we expect the model to be constantly updated on new live data.
  + Component integration: We want to make sure the different components of a system work well together, behave as expected, and communicate with each other using the expected protocols and contracts. We also want to make sure that the behavior of the components in production is the same as the ones in development, i.e., validat- ing with the same datasets should get us the same results. This is not as obvious as it seems, and any mismatch should be investigated.
  + Model quality: The performance of an ML model is non-deterministic, but we can collect and monitor the relevant metrics to evaluate the model’s performance. We will delve into this more, but in general we want to make sure that the relevant met- rics stay within an acceptable range, above all in case of retraining.
  + Bias and fairness: We want to make sure that the model behaves in a way that is aligned with the goal of the organization. This is out of scope from the present topic, though.

###### Monitoring and Observability

Once the model is live, we need to understand how it performs in production, and close the feedback loop to the development process. This is the moment we can gain reliable data on what we have developed so far. The main tools are the standard tools for production software monitoring, with a speciﬁc application to the machine learning process. Production software uses tools for log aggregation and metric collection, to capture the data from a live production system: KPI, software reliability, performance, debugging information in the case of faults, and other indicators that something unex- pected is happening. In general, we want to be notiﬁed when something goes wrong or when something strange happens to allow us to investigate. In a machine learning sys- tem in production, we want to speciﬁcally keep track of

* + - model inputs. These are the data that is being sent to the models, to allow us to keep track of the model-serving skew.
    - model outputs. Seeing what predictions, recommendations, results are the models giving to these inputs, to understand how the model is behaving on real data. This can include the decisions taken based on the outputs, since in some cases we want to deﬁne exceptions where our system will overrule the outputs of the model in some speciﬁc cases.
    - model interpretability outputs. These metrics allow further investigation to under- stand how the models are making predictions to identify potential overﬁt or bias not found during training. As an example, Amazon introduced a machine learning tool to screen the resumes of applicants, that seemed to work quite well, until they realized it was strongly biased against women, since in the training dataset, very few women were hired. Amazon has since stopped using the tool (Dastin, 2018).
    - user action and rewards. We want to keep track of what the users do once they receive the outputs. Do they buy what was presented to them, do they watch the movie, do they pay on time when given credit? This is very important to understand if we are actually solving the right problem.
    - model fairness. As noted above, we want to keep track of the behavior of models regarding race, gender, age, etc. We want to know if the model is behaving in a way that is not aligned to our values.
    - model computational performance. As for all software systems, the reaction time can be critical. For some user interaction, a result delay of a few seconds is the same as no result at all. We want to keep track of how long models take to react, the CPU, and memory load in case we need to scale.

Humans in the loop

(HITL)

This is a model that requires human interaction, as opposed to a closed-loop model that is without human interaction.

Model drift This is the degrada- tion of the perform- ance of the model due to either change in the nature of the current input data or changes in the train-

ing data.

Collecting, monitoring, and observing data is even more important in the case of multi- ple models deployed in production. To assess a shadow model, we can perform A/B tests or run multiple experiments. The collected data is essential to close the data feedback loop: Using more real-world data or adding humans in the loop (HITL) to ana- lyze the new data resulting from production usage, we can create new datasets to gen- erate new, and hopefully improved, models. It allows us to learn to adapt our models based on their behavior on real production data, allowing for continuous improvement.

###### Model Drift

Data can change over time, making the performance of our models degrade if they do not change with the data. We call this problem model drift (Brownlee, 2019).

We can identify two basic kinds of model drift:

* Concept drift: happens when the statistical properties of the desired output itself change. As an example, the predicted spending of customers won’t be accurate if there are promotions not included in the model (Wikipedia, 2020d).
* Data drift: occurs when the statistical properties of the input change, compared to the data used for training. A simple example with seasons: a model of beverage sales trained in the summer will not work without changes in the winter.

Vom Modell zur Produktion

The change in data can take different forms, and we can conceptually model it as

* + gradual change over time. Some trend affects the data, for example people moving to cities from the countryside.
  + recurring or cyclical change. Seasons and days of the week are a typical example, as are holiday season, end of quarter, etc.
  + sudden or abrupt change. Think of the impact of the Covid-19 pandemic on ﬂights and the economy in general, or of the introduction of the iPhone on the sales of other kind of smartphones.

To address model drift, we can consider severalaspects (Zliobaite, 2010).

* + Future assumption: Assumptions about the future data sources, in what way they can differ from what we have, and how to both acquire data and integrate it in the system
  + Change type: Identifying possible change patterns, what changes we can predict, and how we plan to adapt to them. Seasonality is the easiest to think about, under- lying trends due to unknown causes the hardest.
  + Learner adaptivity: Choosing how to adapt the models based on the change type and the future assumption
  + Model selection: A criterion to choose a particular parametrization of the selected learner at different time steps

Possibilities of action and reaction to model drift include

* + doing nothing: We assume that the data does not change and keep a static model. At times, doing nothing performs better than doing something.
  + periodically re-training: We can periodically update your static model with more recent historical data. We can update the model every month with new data, then use the new model as a static model. In some cases, we can use only a small sam- ple of new data, or a sliding window.
  + weighing data: Some algorithms allow us to weigh the importance of input data, so that we can give more weight to more recent data.
  + detecting and choosing a model: In some cases, it is possible to design systems that detect changes and choose a speciﬁc, different model to make predictions. This can be appropriate in systems designed to properly react to sudden changes without human intervention. Consider the case of a trading algorithm that needs to react to a crisis without being rewritten.
  + data preparation: In cases of time series problems, the data are already expected to change over time, and can be partially prepared to remove the systematic changes over time, if those changes can be modelled. Technically, systematic change is not really a model drift, since we can plan for it in advance, but some changes are not systematic, a challenge in using models in the ﬁnancial and other industries.

Seasonality

This is the presence of variations that occur at speciﬁc reg- ular intervals less than a year, such as weekly, monthly, or quarterly.

###### When Things Go Wrong

Now that we have outlined the steps to be considered when bringing a model to pro- duction and the potential challenges, let’s review the reason behind all of these steps: We are trying to minimize problems, and make it easy to recover when things go wrong. In production, something will always go wrong, sooner or later, and we need to both minimize the effects and be able to recover. What can happen, and how can the ele- ments outlined above help? Well, if we stay in production long enough, we will have to retrain the model because of model drift, new challenges, changing business require- ments, new data, or other unforeseen events. We need versioning for this, and we need to make sure that the version includes the data used to train the models. Due to out- ages, or malicious attacks, part of the infrastructure can get lost, but we still need to be able to regenerate the models and we need to make sure the retrained model behaves as we expect. To know when to implement the measures we just learned about, we need monitoring, and also to know when we need more resources because the per- formance is too low. After updating a model, we need to be able to know if the new model behaves as expected (monitoring) and to be able to quickly roll back to a previ- ous version in case it doesn’t (versioning). In the case of online learning (a constantly updated and retrained model), we need to make sure that the performance does not degrade, and we need to be able to roll back to a known working version in case it hap- pens: again, monitoring, versioning, reproducibility.

### MLOps and DataOps

As we have seen in the previous section, putting and keeping a machine learning sys- tem, including the model, in production presents several challenges and is a natural counterpart in the domain of data science to the DevOps of traditional software devel- opment. DevOps is commonly used in the development and operation of large-scale software systems, providing beneﬁts such as shorter development cycles, increased speed of deployment, and dependable releases. The two main concepts used to ach- ieve this are continuous integration (CI) and continuous delivery (CD). A machine learn- ing system is a software system, meaning we can adapt these techniques using the fol- lowing criteria.

###### Team Skills

Data scientists are part of the team and are not always experienced engineers, so their code often needs some work to reach production quality.

Vom Modell zur Produktion

###### Development

Machine learning is experimental and non-deterministic. It is necessary to experiment with different features, algorithms, conﬁgurations, datasets, and problem deﬁnitions to ﬁnd out what works as quickly as possible. It is challenging to keep track of what did or did not work and simultaneously maintaining reproducibility and code reusability.

###### Testen

As noted above, testing an ML system includes elements not present in other software systems. In addition to typical unit and integration tests, you need data validation, trained model quality evaluation, and model validation.

###### Deployment

Machine learning systems can require deployment in a multi-step pipeline to automati- cally retrain and deploy models, adding complexity and the necessity to automate steps that before deployment are manually performed by data scientists.

###### Production

The performance of machine learning systems can be affected by the quality of coding and changes in data proﬁles. Models can decay in more ways than other software sys- tems, and it is necessary to keep track of this with statistics and monitoring, sending notiﬁcations or automatically rolling back to known working versions.

As such, machine learning and other software systems are similar in the requirements for continuous integration of source control, unit testing, integration testing, and con- tinuous delivery of the software module or the package. However, they differ in a few areas. CI is no longer only about testing and validating code and components, but also testing and validating data, data schemas, and models. At the same time, CD is no lon- ger about a single software package or a service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service). We can also add the property of continuous training (CT). CT is a new property, unique to ML systems, that’s concerned with automatically retraining and serving the models (Goo- gle, 2020a). These different and added requirements are leading the creation of two additional practices for data intensive operations: DataOps and MLOps. DataOps (a compound of “data (analytics)” and “operations”) is the practice of using automated, process-oriented methodologies in data teams to improve the quality and reduce the cycle time of data analytics, covering the entire data lifecycle from data preparation to reporting. Toph Whitmore, principal analyst at Blue Hill Research, deﬁned it as follows (Vorhies, 2017):

1. Measure progress and performance at every step of the data ﬂow.
2. Deﬁne rules about what the data and metadata are.
3. Have humans in the feedback loop to make sure assumptions hold.
4. Automate as many stages as possible.
5. Identify bottlenecks.
6. Include governance compliance in the process, including data control, data owner- ship, transparency, and tracking.
7. Design for growth and extensibility, accommodating for increasing volume and vari- ety of data.

MLOps (a compound of “machine learning” and “operations”) is a practice for collabo- ration and communication between data scientists and operations professionals to help manage the production machine learning lifecycle described in the previous sec- tion, covering

* + - deployment and automation,
    - reproducibility of models and predictions,
    - diagnostics,
    - governance and regulatory compliance,
    - scalability,
    - collaboration,
    - business uses, and
    - monitoring and management.

Other names and proposed concepts are AIOps, ModelOps, and DLOps, which are respectively responsible for using AI to automate IT operations, automating ML models, and MLOps speciﬁcally for deep learning. They can all be considered subcategories of MLOps. Given the recent explosion of interest in and usage of machine learning sys- tems, several tools have been developed to support MLOps, including Airﬂow, Luigi, Argo, Kubeﬂow, MLFlow, Michelangelo, and many others. They partly overlap in features, partly solve different problems in different ways, and partly impose a way of solving problems that can either match well with the workﬂow of our organization or present some mismatch. We will cover in depth MLﬂow for model tracking, Kubeﬂow for machine learning pipelines, and Michelangelo for the complete machine learning end- to-end process. These solutions integrate into the existing infrastructure of the organi- zation, in some cases with only small changes and in others, with adaptations to the implicit model of the tools.

###### Model Tracking With MLFlow

MLFlow allows the automation of model development and tracking so that the optimal model can be selected (MLﬂow, n.d.). We can log parameters, attributes, and perfor- mance metrics, using them to identify the models that ﬁt particular criteria. Its main feature is that it enables you to keep track of your ML experiments, amongst others, by logging parameters, results, models, and data for each trial. MLﬂow is designed to be agnostic regarding machine learning libraries, algorithms, deployment tools, or langua- ges. It is aimed to be easily added to an existing machine learning code, sharing code

Vom Modell zur Produktion

inside the organization using any machine learning library, allowing others to run it. While mostly aimed at the model selection part of the lifecycle, it can support the com- plete machine learning lifecycle when used with other tools like Airﬂow. The design philosophy of MLﬂow has a modular and API-based design, with functionality divided into four parts: tracking, projects, models, and registry. MLﬂow tracking is a centralized place for obtaining the details of the model, a sort of meta-store. The client applicati- ons communicate with the tracking server via the HTTP protocol, with APIs in Python, REST, R, and Java. The tracking server captures details for the model and uses backend stores to log

* + logging parameters,
  + code versions,
  + metrics,
  + artifacts (model and data ﬁles),
  + start and end time of the run, and
  + tags and notes.

MLﬂow project

MLﬂow project is organized and packaged code to support the reproducibility of a model. It uses a YAML ﬁle, named MLProject, that describes the requirements of the machine learning project.

Airﬂow

Apache Airﬂow is an open-source work- ﬂow management platform.

A simple example of an MLProject 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 this example, we are deﬁning a decision tree by

1. setting the name to sklearn-demo.
2. declaring we’ll use the Anaconda environment conda.yaml.
3. deﬁning some parameters: max\_depth, required, the maximum depth of the tree, as an integer; max\_leaf\_nodes, an integer with 32 as default value, the maximum num- ber of leaf nodes of the tree; model\_name, a string with the name of the model; run\_origin, where we want to start.
4. the command that will be run, the Python interpreter on the model\_run.py ﬁle.

MLﬂow models

MLﬂow models deﬁne a format for packaging machine learning models that can be used in a variety of downstream tools, like a REST API or Apache Spark. The format allows saving the model using different “ﬂavors” that correspond to different

Anaconda

It is a distribution of the Python and R programming lang- uages for scientiﬁc computing.

tools. “Flavors” are a key concept for MLFlow, a convention to allow deployment tools to understand the models, making it possible to work with any ML library without having to integrate the tools in the library itself. They allow us to

* + utilize the same memory format for different systems,
  + avoid the overhead of cross-system communication (serialization and deserializa- tion), and
  + provide common shareable functionalities.

MLFlow has built-in ﬂavors for many popular machine learning algorithms and libra- ries, such as H2O, Keras, MLeap, PyTorch, Scikit-Learn, MLlib, Tensorﬂow, ONNX (Open Neural Network Exchange), MXNET gluon, XGBoost, and LightGBM. Additionally, it offers the possibility to create custom ﬂavors using Python (MLﬂow, n.d.).

Model registry

The model registry is aimed at solving the problem of machine learning model management, allowing management of the full lifecycle of the machine learning model providing, using the following concepts.

* + Model: A model is created from an experiment or run that logged via speciﬁc MLﬂow logging methods. After being logged, the model can then be registered with the model registry.
  + Registered model: It is then registered in the model registry with a unique name, container versions, associated transitional stages, model lineage, and other meta- data.
  + Model version: Registered models can have one or many versions. When added, a model starts at version 1, with every new model registered to the same model incre- menting the version number.
  + Model stage: Every distinct model version can be assigned to one stage, like staging, production or archived, or something custom. Models can be transitioned to diffe- rent stages.
  + Annotations and descriptions: It is possible to annotate the whole model, and each version, including the description and any relevant information.

MLFlow can be simply be installed as such:

pip install mlflow

A simple pipeline would look like this:

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)

Vom Modell zur Produktion

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")

The code above

* + logs a parameter param1 with value 5;
  + logs metrics called **foo** with values 1, 2, and 3;
  + opens an existing ﬁle (an artifact), output.txt, to write to it; and
  + logs the artifact for the run with log\_artifact. This is what we mostly care about, keeping track of the changes for every time the job runs.

This is clearly a useless example, hence the “Hello, World!”, to just give an idea of how to programmatically annotate the steps of the process in a Python script (GitHub, n.d.- b).

###### Pipeline Management With Kubeﬂow

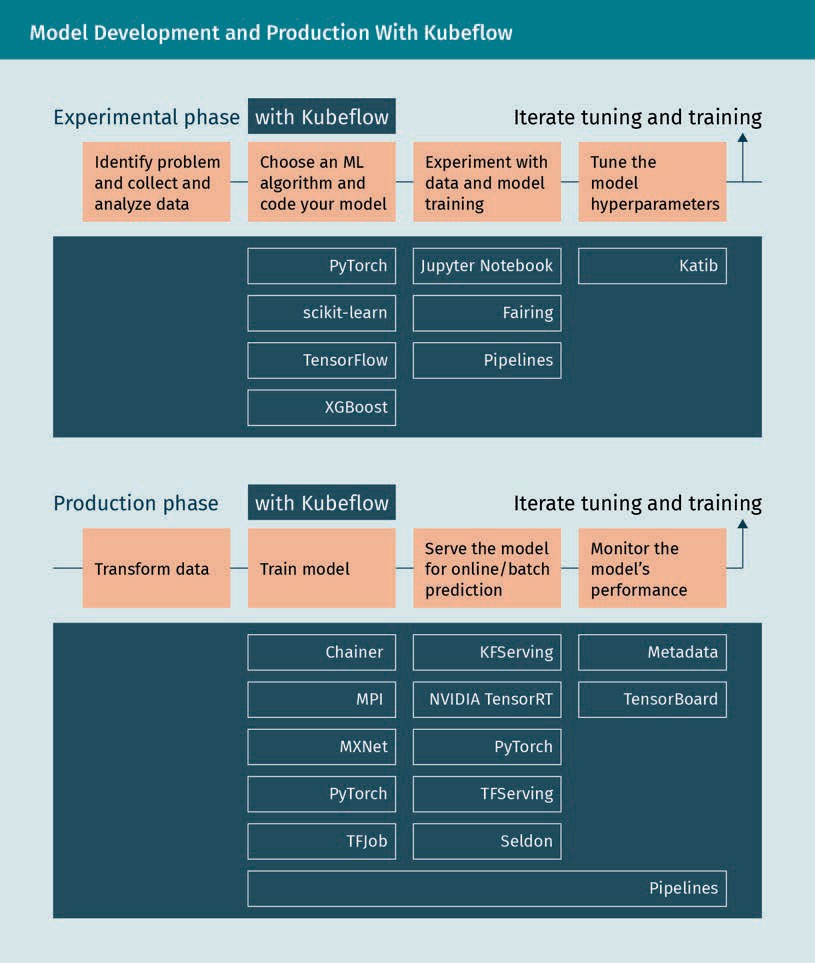
Kubeﬂow was started by Google as an open-source platform for running TensorFlow. It is a machine learning platform that manages deployments of ML workﬂows on Kuber- netes and is a scalable and portable solution (Kubeﬂow. n.d.-a). It is aimed at data scientists building and experimenting with data pipelines, and for deploying machine learning systems to different environments in order to carry out testing, development, and production-level service. It is a multi-cloud, multi-architecture framework that runs entire ML pipelines, and offers speciﬁc tools for all the steps we covered in the model creation pipeline, as well as the deployment pipeline.

Foo

Foobar, foo, bar, and similar are used as placeholder names in computer pro- gramming.

“Hello, World!”

This is a very simple program often used to illustrate the basic syntax of a programming lang- uage.



As you can see from the diagram, Kubeﬂow integrates in commonly used tools (PyTorch, scikit-learn, Jupyter Notebooks, TensorBoard, etc.) and adds some components speciﬁc to KubeFlow. One component is the services for spawning and managing Jupyter Not- ebooks, interactive data science, and experimenting with ML workﬂows. It permits the sharing of notebooks across the organization and the creation of notebooks in pods or containers in the cluster, instead of locally. Kubeﬂow integrates admin controls to allow for standard notebook images and to set up role-based access control (RBAC), as well as secrets and credentials to manage which teams and individuals can access the not- ebooks. Kubeﬂow Pipelines offers a platform for building, deploying, and managing multi-step ML workﬂows based on Docker containers. In Kubeﬂow, a pipeline is a desc- ription of an ML workﬂow, including all components and how they combine. It includes the deﬁnition of inputs and outputs of each component. Every component of a pipeline

Vom Modell zur Produktion

is a Docker image with self-containing code that performs one step of the pipeline. A component can be responsible for data preprocessing, data transformation, model trai- ning, and so on (MLﬂow, n.d.).

The pipelines are described with Python code. What follows is a toy pipeline for a func- tion adding two numbers. Please note that the Python SDK for Kubeﬂow is kfp. We deﬁne a simple function add, we add it to a component, we then create a pipeline cal- led calc\_pipeline (using the @dsl.pipeline decorator), set the arguments we will pass to the pipeline (hence to the function), and ﬁnally create a run for the pipeline with the following 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 offers several components that you can use to build your ML training, hyper- parameter tuning, and serving workloads across multiple platforms. As we can see, it offers solutions to many of the issues we discussed in the previous section.

###### End-to-End Machine Learning Lifecycle With Michelangelo

Michelangelo (Hermann & Del Balso, 2017) was built in 2015 by Uber to tackle what they named “hidden technical debt in machine learning systems” (Sculley et al., 2015). It was built to solve the problem of custom one-off systems tightly coupled with machine learning models, an approach that did not scale well, as we have previously seen.

Technical debt It reﬂects the implied cost of addi- tional rework caused by choosing an easy and limited solution as opposed to a more sustainable approach that would take longer to imple-

ment.

Spark Apache Spark is an open-source distrib- uted general-pur- pose cluster-com- puting framework.

Hadoop Apache Hadoop is a collection of open- source software util- ities that facilitates using a network of many computers to solve problems, involving massive amounts of data and

computation.

Service-level agree-

ment (SLA) This is a commit- ment between a service provider and

a client.

Michelangelo relies on transactional and logged data and supports ofﬂine (batch) and online (streaming) predictions. We use ofﬂine predictions to run the model on a whole dataset at once, getting all results in batch when done, and we use online predictions to respond to queries as they come. Ofﬂine predictions are based on containerized Spark jobs for batch predictions, while online predictions are served in a prediction cluster, allowing for a load balancer to distribute the load to several machines. Load balancing is the process of distributing a set of tasks over a set of resources, with the aim of making their overall processing more efﬁcient. Every experiment stores meta- data relevant to model management (e.g., run-time statistics of the trainer, model con- ﬁguration, lineage, distribution and relative importance of features, model evaluation metrics, standard evaluation charts, learned parameter values, and summary statistics). With Michelangelo it is possible to deploy multiple models in the same serving con- tainer, allowing to transition from old to new model versions and side-by-side A/B test- ing of models. Michelangelo supports both online and ofﬂine models. The data prepa- ration pipelines push data into the feature store tables and training data repositories. They use Apache Kafka (an open-source stream processing software) and connect to a Hadoop File System data lake. The data preparation uses either Apache Spark or SQL. The model training happens in batch, using one of the many supported models: decision trees, linear and logistic models, unsupervised models (k-means), time series models, and deep neural networks. A model conﬁguration speciﬁes the model type, hyperparameters, and data source reference, and computes resource requirements. It is then run on a cluster. After the model is trained, performance metrics are saved into a model evaluation report and the original conﬁguration, the learned parameters, and the evaluation report are saved back to our model repository for analysis and deploy- ment. Hyperparameter search is supported for all model types. For every trained model, a versioned object is stored in the model repository in Cassandra (a distributed, wide column store, NoSQL database management system) for evaluation. Reports can be visualized in a dashboard, with standard accuracy metrics and feature reports. Mod- els can then be deployed in several ways: ofﬂine; in a container then run in a Spark job to generate batch predictions; as an online prediction service cluster; or embedded as a library embedded in another service. Once models are deployed and loaded by the serving container, they are used to make predictions based on feature data loaded from a data pipeline or directly from a client service. Multiple models can be deployed to a given container, and monitoring of metrics and performance is logged. The plat- form uses Spark’s machine learning pipeline serialization with an additional interface for online serving that adds a single-example (online) scoring method, lightweight and capable of handling service-level agreements (SLA), necessary in critical cases such as for fraud detection and prevention.

###### Concluding Remarks

We have covered the challenges of bringing machine learning to production from the point of view of MLOps. The tools we have explored have different strengths and differ- ent approaches. MLFlow, for example, is tightly integrated with Spark (it is developed by the same company). It is mainly a model management library, with limited options for deployment, which needs to happen using Airﬂow. Kubeﬂow is a more integrated solu- tion, build on Kubernetes and integrating several open-source building blocks. It

Vom Modell zur Produktion

requires separate conﬁguration of many of the elements needed. Lastly, Michelangelo is an example of an end-to-end solution, integrating most of the components with structured access to the ones that are not integrated. As such, it imposes some choices on us, removing some ﬂexibility and adding complexity to the system. Each of these systems requires the organization to set up the necessary resources, both hardware and software, and requires an experienced support team.

### Cloud-Based Solutions

We have already discussed some of the challenges posed by bringing machine learning to production, the emergence of MLOps, and tools that solve some, most, or all of these challenges. These tools assume the use of an already existing infrastructure in the organization, and in some cases permit the use of hybrid solutions, accessing cloud infrastructure for some tasks. Building and maintaining an own infrastructure presents some problems and costs. Among others, it requires in-house knowledge, utilizes inter- nal resources, and requires maintenance of the infrastructure on the hardware and software levels, as well as scalability. Some organizations do not want to incur those costs, and the emergence of machine leaning cloud services addresses this need. Using machine learning cloud services, it is possible to build, monitor, and deploy working models, and use them for predictions and insight, all with a relatively small team.

###### Machine Learning as a Service (MLaaS)

Machine learning as a service (MLaaS) is an umbrella deﬁnition of various cloud-based platforms that cover most infrastructure issues we considered for MLOps: data pre-pro- cessing, model training, model evaluation, deployment, and monitoring. The output can then be bridged to the internal IT infrastructure through REST APIs and other forms of internal communication like RPC and other protocols. In 2020, the leading cloud MLaaS providers are Amazon Machine Learning services, Azure Machine Learning, Google Cloud AI, and IBM Watson (IBM, n.d.-c), all aiming to allow for faster model training and deployment (AltexSoft, 2019).

Amazon Machine Learning services are available on two levels: predictive analytics with Amazon ML, and the SageMaker tool mostly aimed at data scientists (Amazon, n.d.-b). Amazon Machine Learning for predictive analytics is a mostly automated solution, aimed at deadline-sensitive operations. It can load data from multiple sources, for example, Amazon RDS, Amazon Redshift, and CSV ﬁles. Data preprocessing operations are performed automatically. Amazon ML tries to identify which ﬁelds are categorical and which are numerical, and it doesn’t ask a user to choose the methods of further data preprocessing such as dimensionality reduction. However, the prediction capaci- ties of Amazon ML are limited to binary classiﬁcation, multiclass classiﬁcation, and regression. It does not support unsupervised learning methods, and a user must select a target variable to label it in a training set. Since it is aimed at non-data scientists, a user isn’t required to know any machine learning methods; instead, they are chosen automatically from the provided data. It provides a fully automated, yet limited, solu-

Cloud infrastructure Cloud computing is the on-demand availability of com- puter system resour- ces, especially data storage and comput- ing power, without direct active man- agement by the user.

Remote procedure call (RPC)

This is when a com- puter program cau- ses a procedure to execute in a differ- ent computer in a network.

tion. For more advanced usage, more ﬂexibility and complexity, there’s SageMaker. Ama- zon SageMaker is a machine learning environment aimed at providing tools for quick model building and deployment, integrating Jupyter for data exploration and analysis without server management hassle (Amazon, n.d.-b).

###### Machine Learning With Amazon SageMaker

To explore how cloud solutions can provide all the tools to build an end-to-end pipe- line for machine learning, we will now concentrate on Amazon SageMaker. The offerings from other providers tend to be similar in their capabilities, with differences in the spe- ciﬁcs. Amazon SageMaker is a fully managed service that provides all the tools to build, train, and deploy machine learning models. SageMaker allows us to execute all the steps of the machine learning pipeline (Amazon, n.d.-c). Let’s quickly review them: understanding the problem, collecting and preparing the data, performing exploratory data analysis, experimenting with features and models, training and selecting the ﬁnal model, keep versions, deploy the model, monitor performance, and integrate with other systems.

Prerequisites

Based on Amazon AWS, using SageMaker requires having or setting up an AWS account, then creating an IAM administrator user and group (Amazon, n.d.-d). AWS is very power- ful and not always intuitive, and it can take some time to ﬁnd out how to do things pro- perly. Bigger organizations will have dedicated processes and teams for this. But if we are setting it up ourselves, we need to make sure to have an IAM user with administra- tive rights. Managing AWS services is a topic that could ﬁll whole books, so it is highly recommended to get at least a bit familiar with it (Amazon, n.d.-e).

Problem understanding

At the moment, no automated system can really help us with problem understanding. That would be an application for a quite advanced AI. SageMaker offers some automa- tic tools, like SageMaker Autopilot, but they are limited to few domains: autopilot, regression, binary classiﬁcation, and multiclass classiﬁcation.

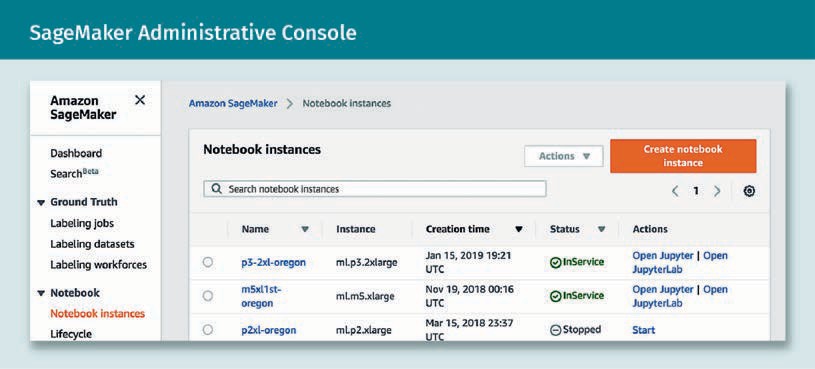
Data lake A data lake is a sys- tem or repository of data stored in its natural (raw) format, usually object blobs

or ﬁles.

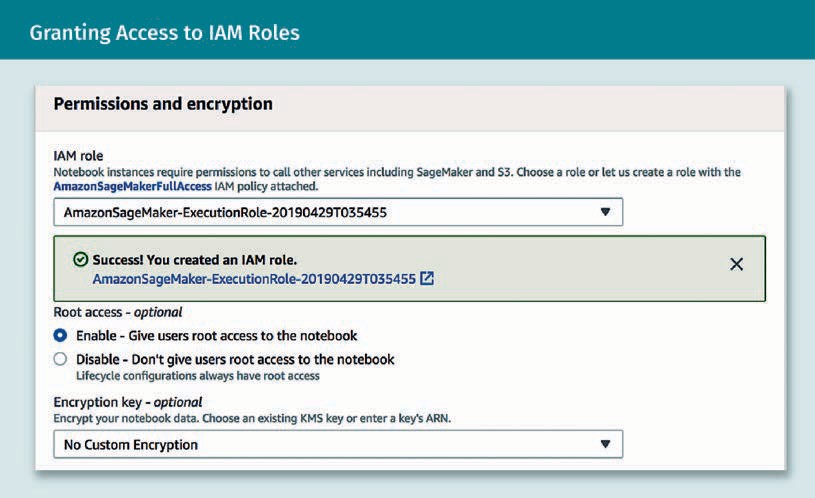
Data collection, cleaning and wrangling

A step we have not considered so far, which is essential when dealing with data in an organization, is where the data come from and where they are stored. As a fully mana- ged solution based on the many services present in AWS, SageMaker offers or integra- tes functionalities for data storage: this step involves setting up a data lake. Generally, we can access data from several sources, but in the AWS environment, most of the time the raw data will be initially stored to an S3 bucket. Amazon Simple Storage Service (S3) provides object storage through a web service interface. We can also save preproces- sing code on S3 and point SageMaker to it. In SageMaker, the data is preprocessed in a Jupyter notebook, in a notebook instance. We use notebooks to fetch the dataset, explore it, and prepare it for model training. To start, we need to launch a notebook instance in SageMaker.

Vom Modell zur Produktion



We can then select name, type of instance, and other settings. In AWS, it is always necessary to create an IAM role. In our speciﬁc case, it is necessary to grant some level of access to the notebook.



Once set up, we can access the notebook instance and open Jupyter. From the note- book, we can perform all the data preprocessing steps, using AWS Glue for executing a SparkML feature pre-processing and post-processing job. AWS Glue is a serverless data preparation service enabling us to extract, clean, enrich, normalize, and load data, and allowing for ETL operations in a graphical interface. SparkML is a high-level API that helps users create and tune practical machine learning pipelines from the Apache Spark project. We can set up several steps for data preprocessing, programmatically or manually. The standard Python client for AWS services is called boto (boto, 2020). The following code, job creation omitted, shows how to run a job using AWS from a note- book:

Jupyter

This is an open- source web applica- tion that allows you to create and share documents that con- tain live code, equat- ions, visualizations, and narrative text.

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)

Exploratory data analysis (EDA)

Using a normal Jupyter notebook, the EDA steps are the same as would be performed on a local machine or on-premise resources, using the same Python code. This step does not need any adaptation; the only difference is that the notebook you work with is hosted on AWS by SageMaker.

Experimentation, training, and versioning

Where the tools provided by SageMaker begin to shine is when we start experimenting with models, features, and hyperparameters. We run everything as a training job, either from the UI or from the Python SDK. SageMaker includes several built-in models that we can apply to a problem (Amazon, n.d.-f). They cover supervised learning, unsupervi- sed learning, textual analysis, and image processing. The complete list of currently avai- lable algorithms in 2020 is BlazingText algorithm, DeepAR forecasting algorithm, factori- zation machine, image classiﬁcation algorithm, IP insights, k-means algorithm, k- nearest neighbors (k-NN) algorithm, latent dirichlet allocation (LDA) algorithm, linear learner, neural topic model (NTM) algorithm, Object2Vec algorithm, object detection algorithm, principal component analysis, random cut forest (RCF), semantic segmenta- tion algorithm, SageMaker sequence to sequence (seq2seq), and XGBoost (eXtreme gra- dient boosting). It is also possible to use your own algorithm, packaging it in a Docker container, by either extending a prebuilt Docker image or adapting/creating your own. For feature selection and hyperparameter tuning, SageMaker is well integrated with SparkML, using it for most tasks. One of the most important contributions of SageMaker is “Experiments,” letting you organize, track, compare, and evaluate your machine lear- ning experiments. SageMaker experiments introduce the concept of trial and experi- ment. A trial is a collection of training steps involved in a single training job, including preprocessing, training, and evaluation, enriched with metadata. An experiment is a collection of trials, a group of related training jobs. In SageMaker experiments we create experiments, populate them with trials, and run analytics across trials and expe-

Vom Modell zur Produktion

riments. It is available both from the UI and through a Python SDK containing logging and analytics APIs. Additionally, experiments are integrated in Sagemaker autopilot, so that when we perform an autopilot job, an experiment is automatically created with tri- als for the different combinations of components, hyperparameters, and artifacts. Both in training jobs and autopilot, we just have to add an extra parameter to the Python code in the SDK to deﬁne the experiment. The end result is a collection of experiments, updated in real time, that we can compare and evaluate, visually and programmatically, to choose the best model so far. Please note that while experiments permit us to keep a version of our models, we need to take care of data and code versioning ourselves. SageMaker notebooks support Git, and we can develop or integrate data versioning sys- tems in our workﬂow. We can repeat the same process for the ﬁnal training; since we can use the Python SDK, we can automate it programmatically. SageMaker supports additional model validation, with ofﬂine or live data, for A/B testing.

Deployment

Once the selected model is trained, we can deploy it on SageMaker in two different ways. We can set up a persistent endpoint to get one prediction at a time, using Sage- Maker hosting services, or we can get predictions for an entire dataset, using SageMa- ker batch transform. To use the hosting services, we need to create an HTTPS endpoint and pass it to SageMaker. This will allow it to be used for predictions. Note that these endpoints scoped on individual AWS accounts are not public. It is possible to use AWS Lambda and the Amazon API Gateway to allow public access, if desired (Amazon, n.d.-g).

Überwachung

Where the AWS environment really shines is in deployment, production, and monitoring support. SageMaker offers a model monitor tool to check the quality of the models in real time, while the Amazon CloudWatch model monitor enables us to set up an auto- mated alert triggering system when there are deviations in the model quality, such as data drift and anomalies. Amazon CloudWatch Logs collect log ﬁles and notify when the quality of the model hits certain speciﬁed thresholds. AWS CloudTrail stores the log ﬁles to an Amazon S3 bucket (Amazon, n.d.-h). It is possible to detect model deviations and other potential problems early and proactively, countering the problem of model drift. SageMaker also offers functionalities to test multiple models or model versions behind the same endpoint using production variants. It is possible to test them by spe- cifying trafﬁc distribution, invoking speciﬁc variants, and running A/B tests. We can then evaluate the model performance, and increase the trafﬁc reaching the best model. If necessary, we can also trigger retraining of the models, using the same capabilities for the initial training and selection.

###### Concluding Remarks

As we have seen in the example of Amazon SageMaker, cloud providers of machine learning as a service allow us to perform the complete model development and pro- duction pipeline, in some cases as we would use them on premises resources, in other cases, supporting us with several solved problems. Cloud providers are aimed at scala- ble production environments, so they offer out-of-the-box functionality like monitoring and scalability. They allow us to start small, with a small team and few resources, and

Vendor lock-in It makes a customer dependent on a ven- dor for products and services, unable to use another vendor without substantial switching costs.

grow as needed. There are tradeoffs: Even if the cloud providers are often based on open-source software (Jupyter, SparkML, Python, PyTorch, Docker, Kubernetes, Git, to name a few), the providers either modify it, or package it quite tightly with closed- source solutions. While this is not a problem per se, it impacts portability. As an exam- ple (in case we want to move our infrastructure away from SageMaker), if we have used built-in algorithms to train a model (the most straightforward solution), we would then have to reimplement the solution, or rewrite some of the code. Cloud providers also present an additional cost. The management of the infrastructure comes with an addi- tional price tag, for example, a SageMaker EC2 instance on AWS is 40% more expensive than a bare bone AWS EC2 instance. This tradeoff is generally worth it in small organi- zations that do not already have IT support and the necessary resources, but if the lat- ter are present, the cost would just be overhead. Finally, our solutions end up being developed with a speciﬁc architecture in mind, adapting to the assumptions and tools of the provider. This can be for the best, in case we are not experienced enough, since the architectures of the cloud providers are based on best practices, but it comes at the cost of ﬂexibility. This all contributes to vendor lock-in. Our machine learning solu- tion ends up linked to a provider, with potentially high costs to change (which increase as the complexity of our machine learning system increases). In general, as an organi- zation grows, it gets to a point where it makes sense to take care of its own infrastruc- ture. Doing this too early imposes unnecessary costs, while doing it too late presents both costs and risks. To summarize, the beneﬁts of machine learning cloud solutions are

* scalable production environments,
* integrated monitoring,
* implicit best practices, and
* that it solves several of the problems for us.

Alternately, the issues of machine learning cloud solutions can be

* vendor lock-in and portability issues,
* increased costs, and
* closed source solution.

Zusammenfassung

This unit introduced the characteristics of the development process of a machine learning model: exploratory, iterative, and non-deterministic. We explored some challenges related to data quality and exploration, and to the development, testing, and deployment of machine learning models. We then approached the many chal- lenges in bringing and keeping a machine learning system in production—some similar to the ones encountered in the traditional software development process, some slightly different, and some intrinsic to the non-deterministic and data- driven nature of machine learning. We explored how versioning and testing is dif- ferent in the ﬁeld of machine learning, and how to adapt to potential issues. This unit introduced the concept of MLOps, the practice of collaboration between data

Vom Modell zur Produktion

scientists and other members of the production team, and several modern tools offering solutions to some of the challenges. We ﬁnally reviewed machine learning cloud solutions offering end-to-end approaches, as well as their strengths and their weaknesses.



# Anhang 1

## Referenzen

Referenzen

AltexSoft. (2019, September 27). *Comparing machine learning as a service: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson.* https://[www.altexsoft.com/blog/data-](http://www.altexsoft.com/blog/data-) science/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google- cloud-ai-ibm-watson/

Apache Mesos. (o. D.) *What is Mesos? A distributed systems kernel.* http:// mesos.apache.org/

Amazon. (o. D.-a). *What is DevOps?* Amazon Web Services. https://aws.amazon.com/ devops/what-is-devops/

Amazon. (o. D.-b). *Amazon SageMaker.* Amazon Web Services*.* https://aws.amazon.com/ sagemaker/

Amazon. (o. D.-c) *Machine learning on AWS*. Amazon Web Services. https:// aws.amazon.com/machine-learning/

Amazon. (o. D.-d). *Set up Amazon SageMaker*. Amazon Web Services. https:// docs.aws.amazon.com/sagemaker/latest/dg/gs-set-up.html

Amazon. (o. D.-e). *Hands-on tutorials*. Amazon Web Services. https://aws.amazon.com/ getting-started/hands-on/

Amazon. (o. D.-f) *Use Amazon SageMaker built-in algorithms*. Amazon Web Services. https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html

Amazon. (o. D.-g). *Deploy a model 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*. Amazon Web Services. https:// docs.aws.amazon.com/sagemaker/latest/dg/monitoring-overview.html

Anji, K. (2020, September 22). *12 factor app principles and cloud-native microservices*. DZone. https://dzone.com/articles/12-factor-app-principles-and-cloud-native-microser

AOE. (o. D.). *Agile methods & processes in companies*. https://[www.aoe.com/en/](http://www.aoe.com/en/) agile.html

Apache Maven Project. (o. D.). *Welcome to Apache Maven*. Apache. https:// maven.apache.org/

API Evangelist. (2012). *The secret to Amazon's success internal APIs.* https://apievangel- ist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/

Referenzen

Arora, S. (2020, July 1). *Ansible vs. Puppet: The key differences to know.* Simplilearn Solutions. https://[www.simplilearn.com/ansible-vs-puppet-the-key-differences-to-know-](http://www.simplilearn.com/ansible-vs-puppet-the-key-differences-to-know-) article

Atlassian Bitbucket. (o. D.). *Git checkout.* Atlassian. https://[www.atlassian.com/git/tutori-](http://www.atlassian.com/git/tutori-) als/using-branches/git-checkout

AVI Networks. (o. D.). Single point of failure deﬁnition. https://avinetworks.com/glos- sary/single-point-of-failure/

Bach, J. (2003). *Exploratory testing explained.* Satisﬁce. <http://satisﬁce.us/articles/et-> article.pdf

Ballard, G., & Howell, G. (2003). Lean project management. *Building Research & Information*, *31*(2), 119—133.

Baucherel, K. (2019). Scrum in rugby. *Pixabay*. https://pixabay.com/photos/rugby- scrum-heineken-cup-saracens-4498375/

Basu, A. (2015). *Software quality assurance, testing and metrics*. PHI Learning.

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70—77. Beck, K. (2014). *Test-driven development: By example*. Addison-Wesley.

Bentley, C. (2010). *PRINCE2: A practical handbook* (3rd ed.)*.* Routledge.

Berg, J., Graham, M., & Whitney, K. (1981). *Database architectures, a feasibility workshop report.* U.S. Department of Commerce. https://babel.hathitrust.org/cgi/pt? id=mdp.39015077587742&view=1up&seq=53

Berners-Lee, T. (1996). *Hypertext transfer protocol – HTTP/1.0*. https://tools.ietf.org/ html/rfc1945

Bitkeeper. (o. D.). *Bitkeeper homepage.* https://[www.bitkeeper.org](http://www.bitkeeper.org/)

Bourne, S. R. (1978). Unix time-sharing system: The Unix shell. *The Bell System Technical Journal*, *57*(6), 1971—1990.

Boyd, M. (2014, February 21). *Private, partner or public: Which API strategy is best for business?* Programmableweb. https://[www.programmableweb.com/news/private-part-](http://www.programmableweb.com/news/private-part-) ner-or-public-which-api-strategy-best-business/2014/02/21

boto. (2020). *boto: A Python interface to Amazon Web Services*. Cloudhackers. https:// boto.cloudhackers.com/en/latest/

Breck, E., Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2019). Data validation for machine learning*. Proceedings of the 2nd SysML conference*. MLSys.

Bright, P. (2016, August 18*). PowerShell is Microsoft’s latest open source release, coming to Linux, OS X.* Ars Technica*.* https://arstechnica.com/information-technology/2016/08/ powershell-is-microsofts-latest-open-source-release-coming-to-linux-os-x/

Brikman, Y. (2016, September 26). *Why we use Terraform and not Chef, Puppet, Ansi- ble, SaltStack, or 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 mod- els are few-shot learners*. arXiv. https://arxiv.org/abs/2005.14165v4

Brownlee, J. (2019, August 12). *A gentle introduction to concept drift in machine learning*. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction- concept-drift-machine-learning/

Buchanan, C. (2019, June 12). *The ideal DevOps team structure*. GitLab. https:// about.gitlab.com/blog/2019/06/12/devops-team-structure/

Carilli, J. F. (2013). *Transitioning to Agile: Ten success strategies* [Paper presenta- tion]. Project Management Institute Global Congress.

Chappell, D. (2008). *What is application lifecycle management?* Wayback Machine. https://web.archive.org/web/20141207012857[/http://www.microsoft](http://www.microsoft.com/global/appli-).[com/global/appli-](http://www.microsoft.com/global/appli-) cationplatform/en/us/RenderingAssets/Whitepapers/What%20is%20Application

%20Lifecycle%20Management.pdf

Chef. (o. D.). *Chef homepage*. https://[www.chef.io/](http://www.chef.io/)

Chen, L. (2015). Continuous delivery: Huge beneﬁts, but challenges too. *IEEE Software, 32*(2), 50—54.

Chollet, F. (2017). *Deep learning with Python.* Manning. CircleCI. (o. D.). *CircleCI homepage*. https://circleci.com

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

Cockburn, A., & Williams, L. (2001). *The costs and beneﬁts of pair programming: Extreme programming examined*. Addison-Wesley.

Cohn, M. (2009). *Succeeding with Agile: Software development using Scrum*. Pearson Education.

Referenzen

Collado, A. (2018, August 16). *Improving how we work through physical Kanban boards*. Kalibrr Design. https://medium.com/kalibrr-design/improving-how-we-work-through- physical-kanban-boards-74779327bcec

Consultancy. (2020, May 7). *Half of companies applying Agile methodologies & practices*. https://[www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodolo-](http://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodolo-) gies-practices

Cotton, I., & Greatorex, F. (1968). Data structures and techniques for remote computer graphics. *AFIPS joint computer conferences: Fall, part I* (533—544). Association for Com- puting Machinery. https://doi.org/10.1145/1476589.1476661

CruiseControl. (o. D.). *CruiseControl homepage*. [http://cruisecontrol.sourceforge.net](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*. Lulu Publishing Services.

Dastin, J. (2018, October 11). *Amazon scraps secret AI recruiting tool that showed bias against women*. Reuters. https://[www.reuters.com/article/us-amazon-com-jobs-auto-](http://www.reuters.com/article/us-amazon-com-jobs-auto-) mation-insight-idUSKCN1MK08G

Dean, J., & Ghemawat, S. (2004). MapReduce: Simpliﬁed data processing on large clus- ters. *OSDI ’04: 6th symposium on operating systems design and implementation*, *6*. USE- NIX Association.

DevOps. (2015, February 17). *Comparing DevOps to traditional IT: Eight key differen- ces*. 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*. https://docs.docker.com/glossary/ Docker. (n.d.-c). *Docker hub*. https://hub.docker.com/

Docker. (o. D.-d). *Docker swarm*. https://docs.docker.com/engine/swarm

DonWells. (2013). XP-feedback. *Wikimedia Commons*. https://commons.wikimedia.org/ wiki/File:XP-feedback.gif

Dudler, R. (n.d.). *Git: The simple guide*. https://rogerdudler.github.io/git-guide/

DVC. (o. D.). *Open-source version control system for machine learning projects*. Data Ver- sion Control. https://dvc.org/

Dwork, C., Hardt, M., Pitassi, T., Reingold, O., & Zemel, R. (2012). *Fairness through aware- ness.* arXiv*.* https://arxiv.org/abs/1104.3913v2

Eclipse. (o. D.). *Eclipse homepage*. https://[www.eclipse.org](http://www.eclipse.org/)

Elastic. (o. D.). *The ELK stack in a DevOps environment*. https://[www.elastic.co/de/webi-](http://www.elastic.co/de/webi-) nars/elk-stack-devops-environment

Fazliu, A. (2018, June 2). *DevOps: To do or not to do?* Towards data science. https:// towardsdatascience.com/devops-to-do-or-not-to-do-focus-on-culture-ﬁrst- f82319ed346a

Fielding, R. (2000). *Architectural styles and the design of network-based software archi- tectures.* University of California.

Fisher, R. A. (1936). *Iris data set*. UCI Machine Learning Repository. https:// archive.ics.uci.edu/ml/datasets/iris

Fowler, M. (2002). *Public vs. published interfaces.* Martin Fowler. https://martin- fowler.com/ieeeSoftware/published.pdf

Fowler, M. (2009). *TwoHardThings*. Martin Fowler. https://martinfowler.com/bliki/ TwoHardThings.html

Fowler, M., & Beck, K. (1999). *Refactoring: Improving the design of existing code.* Addi- son-Wesley.

Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. *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. https://[www.gnu.org/software/](http://www.gnu.org/software/) wget/

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

Freeman, S., & Pryce, N. (2012). *Growing object-oriented software, guided by tests*. Addi- son-Wesley.

Gade, K. (2019, June 13). *AI needs a new developer stack!* 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*. Git LFS. https://guides.github.com/activities/hello-world/ GitHub. (n.d.-c). *Git large ﬁle storage*. Git LFS. https://git-lfs.github.com/

Referenzen

Google. (2020a). *Classiﬁcation: Precision and recall.* 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*. 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](http://www.gocd.org/)

Golden, B. (2014, February 11). *How DevOps can accelerate the cloud application lifecyle*. CIO. https://[www.cio.com/article/2378823/how-devops-can-accelerate-the-cloud-appli-](http://www.cio.com/article/2378823/how-devops-can-accelerate-the-cloud-appli-) cation-lifecycle.html

Hearn, S. (2019, January 18). *5 great examples of agile organisations*. Clear Review. https://[www.clearreview.com/5-examples-of-agile-organisations/](http://www.clearreview.com/5-examples-of-agile-organisations/)

Heck, P. (2020). *Testing machine learning applications.* Fontys. https://fontysblogt.nl/ testing-machine-learning-applications/

Hermann, J., & Del Balso, M. (2017, September 5). *Meet Michelangelo: Uber’s machine learning platform.* Uber Engineering. https://eng.uber.com/michelangelo-machine- learning-platform/

Highsmith, J. (2009). *Agile project management: Creating innovative products*. Pearson Education.

Humble, J., & Farley, D. (2015). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.

H2O. (o. D.). *Explainable AI*. https://[www.h2o.ai/explainable-ai/](http://www.h2o.ai/explainable-ai/) IBM*.* (o. D.-a) *AI Fairness 360.* [http://aif360.mybluemix.net](http://aif360.mybluemix.net/)

IBM*.* (o. D.-b). *Rational ClearCase*. https://[www.ibm.com/products/rational-clearcase](http://www.ibm.com/products/rational-clearcase)

IBM. (n.d.-c). *IBM Watson machine learning*. https://[www.ibm.com/cloud/machine-](http://www.ibm.com/cloud/machine-) learning

Import.io. (2017, November 17). *Is web scraping legal? 6 misunderstandings about web scraping*. https://[www.import.io/post/6-misunderstandings-about-web-scraping/](http://www.import.io/post/6-misunderstandings-about-web-scraping/)

Jenkins. (o. D.-a) *Jenkins: Build great things at any scale*. https://[www.jenkins.io/](http://www.jenkins.io/)

Jenkins. (o. D.-b). *Build a Java app with Maven*. https://[www.jenkins.io/doc/tutorials/](http://www.jenkins.io/doc/tutorials/) build-a-java-app-with-maven/

Jenkins. (n.d.-c). *Using a Jenkinsﬁle.* https://[www.jenkins.io/doc/book/pipeline/jenkins-](http://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*. BIS Publishers.

Jupyter Team. (o. D.). *The 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.* Flor- ida Institute of Technology. <http://www.testingeducation.org/k04/OracleExamples.htm>

Karpathy, A. (2017, November 11). *Software 2.0*. Medium. [https://medium.com/@karpa-](https://medium.com/%40karpa-) thy/software-2-0-a64152b37c35

Katalon. (2020). *What is end-to-end (E2E) testing? All you need to know.* https:// [www.katalon.com/resources-center/blog/end-to-end-e2e-testing/](http://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. *IEE Transactions on Software Engineering, 44*(11), 1024— 1038.

Kubeﬂow. (o. D.-a) *Kubeﬂow: An introduction to Kubeﬂow*. https://[www.kubeﬂow.org/](http://www.kubeﬂow.org/) docs/about/kubeﬂow/

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

Kubernetes. (n.d). *Kubernetes concepts*. https://kubernetes.io/docs/concepts/ Kumar, R. (2016). *Human computer interaction*. Firewall Media.

Kwak, Y. (2005). A brief history of project management. In E. G. Carayannis, F. T. Anbari, &

Y. Kwak (Eds.), *The story of managing projects* (pp. 1—9). Praeger.

Lane, K. (2019, October 10). Intro to APIs: History of APIs. *Postman*. https://blog.post- man.com/intro-to-apis-history-of-apis/

Lumen. (2020) *Introduction to computer applications and concepts*. https://cour- ses.lumenlearning.com/computerapps/chapter/reading-the-internet/

Malamud, C. (1990). *Analyzing novell networks*. Van Nostrand Reinhold.

Mangat, M. (2019, November 4). *Kubernetes vs. Docker Swarm: What are the differen- ces?* 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. *The International Journal of Logistics Manage- ment, 18*(2), 238—254.

Referenzen

Mehrjerdi, Y. Z. (2011). Six-Sigma: Methodology, tools and its future. *Assembly Automa- tion*, *31*(1), 79—88. <http://dx.doi.org/10.1108/01445151111104209>

Mercurial. (o. D.). *Mercurial homepage.* https://[www.mercurial-scm.org/](http://www.mercurial-scm.org/)

Misa, T., & Phillip, F. (2010*).* An interview with Edsger W. Dijkstra. *Communications of the ACM, Vol. 53*(8). https://doi.org/10.1145/1787234.1787249

Mitchell, I. (2015). Scrum Framework. *Wikimedia Commons*. https://commons.wikime- dia.org/wiki/File:Kanban\_principles.jpg

MLﬂow. (o. D.). *Built-in model ﬂavors*. MLﬂow Project. https://[www.mlﬂow.org/docs/](http://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/](http://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 pro- duce successful and quality software. *13th International conference on computational science and its applications* (119—123). IEEE Press.

Newkirk, J., & Martin, R. C. (2000). Extreme programming in practice. *OOPSLA ’00: Adden- dum to the 2000 proceedings of the conference on object-oriented programming, sys- tems, language, and application* (pp. 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. *Proceedings of the international conference on software and system process* (pp. 140—149). IEEE Press.

North, D. (2006). Behavior modiﬁcation, better software. *Stickyminds, 2006*(3). https:// [www.stickyminds.com/better-software-magazine/behavior-modiﬁcation](http://www.stickyminds.com/better-software-magazine/behavior-modiﬁcation)

North, D. (2020). *Introducing BDD*. Dan North & Associates. https://dannorth.net/intro- ducing-bdd/

Null, C. (n.d.). *Infrastructure as code: The engine at the heart of DevOps.* TechBeacon. https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops

Pandas. (o. D.). *Pandas*. PyData. https://pandas.pydata.org/

Patel, C. (2020, July 02). *DevOps best practices*. DZone. https://dzone.com/articles/ devops-best-practices

Perforce. (o. D.). *Perforce homepage*. https://[www.perforce.com](http://www.perforce.com/)

Price, D. R. (2005, April 18). *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)*.

Python Shell. (2020, April 7). *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. *Wikipedia Commons*. https://en.wikipedia.org/wiki/PRINCE2#/media/File:PRINCE2\_-\_Project\_Manage- ment\_Methodology.png

Ré, C., Niu, F., Gudapati, P., & Srisuwananukorn, C. (2019, September). *Overton: A data sys- tem for monitoring and improving machine-learned products*. Apple Machine Learning Research. https://machinelearning.apple.com/research/overton

Red Hat Ansible. (o. D.-a) How Ansible works. Red Hat. https://[www.ansible.com/over-](http://www.ansible.com/over-) view/how-ansible-works

Red Hat Ansible. (o. D.-b). *Automation: Learning Ansible basics.* Red Hat. https:// [www.redhat.com/en/topics/automation/learning-ansible-tutorial](http://www.redhat.com/en/topics/automation/learning-ansible-tutorial)

Red Hat Ansible. (n.d.-c). What is container orchestration? Red Hat. https:// [www.redhat.com/en/topics/containers/what-is-container-orchestration](http://www.redhat.com/en/topics/containers/what-is-container-orchestration)

Rimol, M. (2019, December 10). *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.* Gartner. https://[www.gartner.com/smarterwithgartner/gartner-](http://www.gartner.com/smarterwithgartner/gartner-) top-10-trends-impacting-infrastructure-operations-for-2020/

Rochkind, M. J. (1975). The source code control system. *IEEE Transactions on Software Engineering, 1*(4), 364—370.

Rosenberg, D., & Stephens, M. (2008). *Extreme programming refactored: The case against XP*. Apress.

Rossberg, J., & Olausson, M. (2014). *Beginning application lifecycle management.* Apress.

Sadrach, P. (2020). *Why Keras is the leading deep learning API: Understanding the Keras deep learning library.* 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.* Infowar. https://[www.infoworld.com/article/3271126/what-is-cicd-continu-](http://www.infoworld.com/article/3271126/what-is-cicd-continu-) ous-integration-and-continuous-delivery-explained.html

Referenzen

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. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28 NIPS*. NeurIPS Proceedings. https://papers.nips.cc/ paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

Serrador, P., & Pinto, J. K. (2015). Does Agile work? A quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040—1051.

Shaikh, A. (2017, July 9). *An overview of IT roles, Waterfall and Agile.* LinkedIn. https:// [www.linkedin.com/pulse/department-ayub-shaikh/](http://www.linkedin.com/pulse/department-ayub-shaikh/)

Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile processes and methodologies: A concep- tual study. *International Journal on Computer Science and Engineering, 4*(5), 892.

Siegelaub, J. (2004). How PRINCE2 can complement PMBOK and your PMK. *Project Man- agement Institute Global Congress 2004*.

Sims, C., & Johnson, H. L. (2012). *Scrum: A breathtakingly brief and agile introduc- tion*. Dymaxicon.

Specﬂow. (o. D.). *Gherkin.* https://specﬂow.org/bdd/gherkin/

Spolsky, J. (2002, November 11). *The law of leaky abstractions.* Joel on Software. https:// [www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/](http://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/)

Stallman, R. (2011). *The origin of the name POSIX*. Stallman. https://stallman.org/arti- cles/posix.html

StarTeam. (o. D.). *StarTeam homepage*. Micro Focus. https://[www.microfocus.com/en-us/](http://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. *The International Journal of Production Research, 15*(6), 553—564.

Sutherland, J. (2004). Agile development: Lessons learned from the ﬁrst Scrum. *Cutter Agile Project Management Advisory Service: Executive Update, 5*, 1—4.

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017, May). Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 scientiﬁc workshops, 23*, 1—5.

Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review, 64*(1), 137—146.

Tensorﬂow. (o. D.). *Tensorﬂow homepage*. https://[www.tensorﬂow.org](http://www.tensorﬂow.org/)

Terraform. (o. D.). *Terraform documentation*. https://[www.terraform.io/docs/index.html](http://www.terraform.io/docs/index.html)

Thedev, J. (2019, July 18). *The eight phases of a 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 sys- tem. *Proceedings of the 6th international conference on software engineering* (pp. 58– 67)*.* IEEE Press.

Tozzi, C. (2017, May 9). *Understanding why Docker is so popular*. 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*. CRN. https://[www.crn.com/news/mobility/oracle-v-google-copyright-case-slated-for-](http://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.* Browserstack. https:// [www.browserstack.com/guide/tdd-vs-bdd-vs-atdd](http://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd)

University of Maryland. (2018). *Resources for CMSC 341 Data Structures.* https://user- pages.umbc.edu/~park/cs341.f18/resources/

Vargo, S., & Fong-Jones, L. (2018, May 8). *SRE vs. DevOps: Competing standards or close friends?* Google. https://cloud.google.com/blog/products/gcp/sre-vs-devops-compet- ing-standards-or-close-friends

Verona, J. (2016). *Practical DevOps.* Packt Publishing.

Vim. (o. D.). *Vim: the ubiquitous text editor.* https://[www.vim.org](http://www.vim.org/)

Visual Studio Code. (2020, November 6). *IntelliSense*. https://code.visualstudio.com/ docs/editor/intellisense

VMware. (o. D.). *Container orchestration.* https://[www.vmware.com/topics/glossary/](http://www.vmware.com/topics/glossary/) content/container-orchestration

Vorhies, W. (2017, April 4). *DataOps – It’s a secret*. Data Science Central. https:// [www.datasciencecentral.com/proﬁles/blogs/dataops-it-s-a-secret](http://www.datasciencecentral.com/proﬁles/blogs/dataops-it-s-a-secret)

Waldner, J. (1992). Kanban principles. *Wikipedia Commons.* https://commons.wikime- dia.org/wiki/File:Kanban\_principles.svg

Wiggins, A. (2017). *The twelve-factor app*. 12 Factor. https://12factor.net/

Wikipedia. (2020a). *Site reliability engineering*. https://en.wikipedia.org/wiki/Site\_relia- bility\_engineering

Referenzen

Wikipedia. (2020b). *Transformational leadership*. https://en.wikipedia.org/wiki/Transfor- mational\_leadership

Wikipedia. (2020c). *Data wrangling*. https://en.wikipedia.org/wiki/Data\_wrangling Wikipedia. (2020d). *Concept drift*. https://en.wikipedia.org/wiki/Concept\_drift Wikipedia. (2020e). *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.* IEEE Transactions on Software Engineering. https://dx.doi.org/ 10.1109/TSE.2019.2962027

Zliobaite, I. (2010). *Learning under concept drift: An overview*. arXiv. https:// arxiv.org/abs/1010.4784v1

Z Shell. (o. D.) *An Introduction to the Z Shell*. SourceForge. <http://zsh.sourceforge.net/> Intro/intro\_toc.html