KURSBUCH



## Softwareentwicklung für datenintensive Wissenschaften

DLMDSSEDIS01



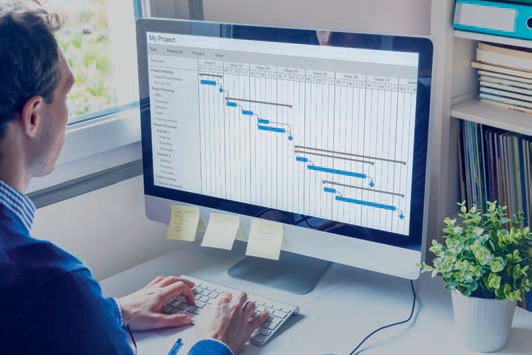
Lernziele

##### 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

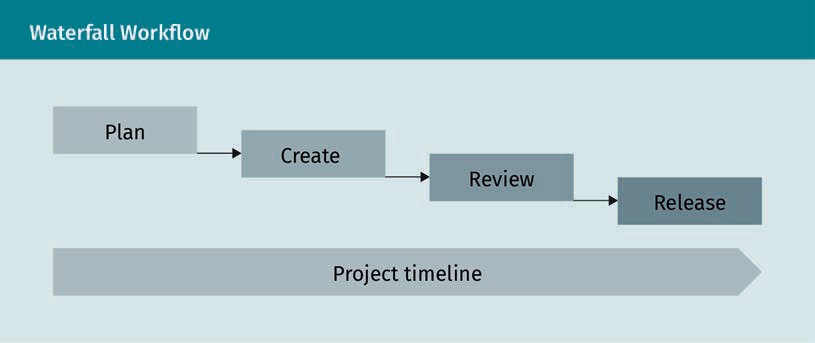
Die Arbeit einer Abteilung oder eines Teams kann in die Bereiche laufender Betrieb und neue Projekte aufgespalten werden. Im laufenden Betrieb werden tägliche Vorgänge abgedeckt. In einer Softwareentwicklungsabteilung eines Autoherstellers ist das Hauptziel beispielsweise das Erstellen der für das Unternehmen 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

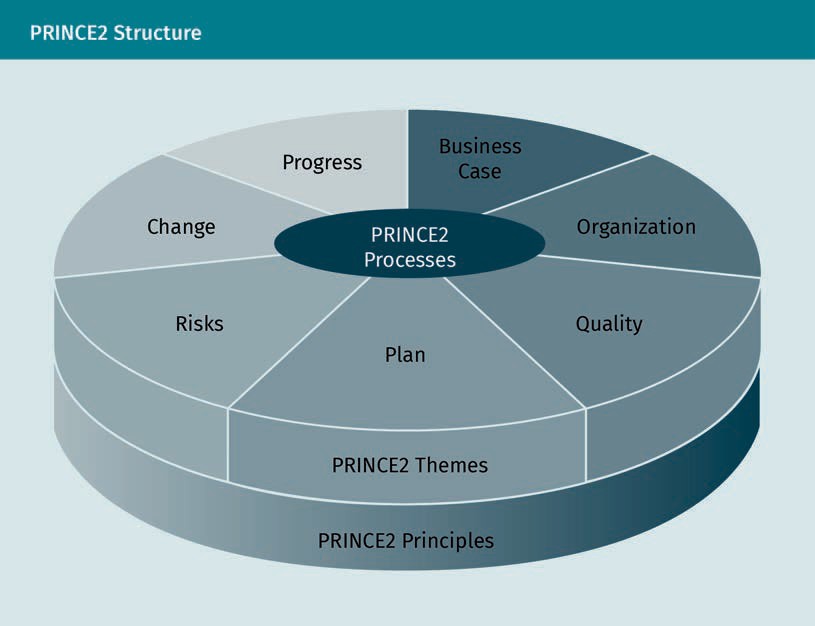
Im Lebenszyklus spielen auch sieben Themen bei PRINCE2 eine Rolle. Sie müssen ständig in die Projektaufgaben und -aktivitäten einfließen:

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.

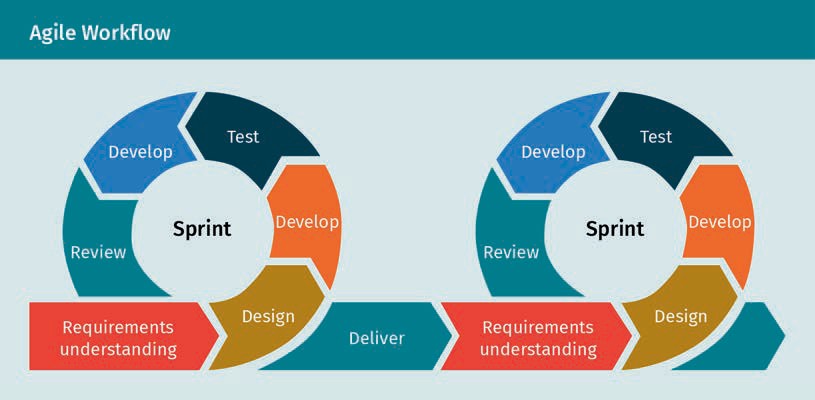
Nach Erläuterung der Werte und Prinzipien des agilen Projektmanagements können wir tiefer in diese Methode einsteigen. Agile ist eine iterative Form des Projektmanagements, die Organisationen dabei unterstützt, Produkte und Dienstleistungen schnell und problemlos an ihre Kunden zu liefern. Die Struktur 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.

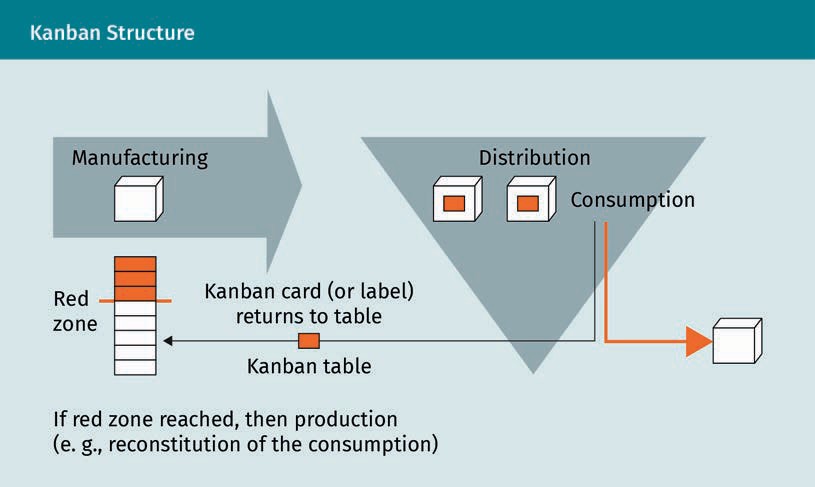
### Kanban

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 gleichbleibender 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:

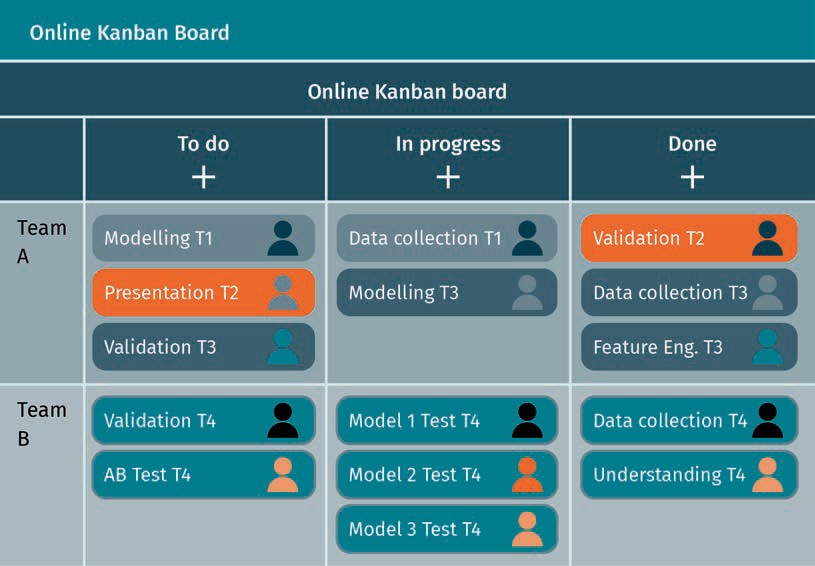
Pull-System

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.

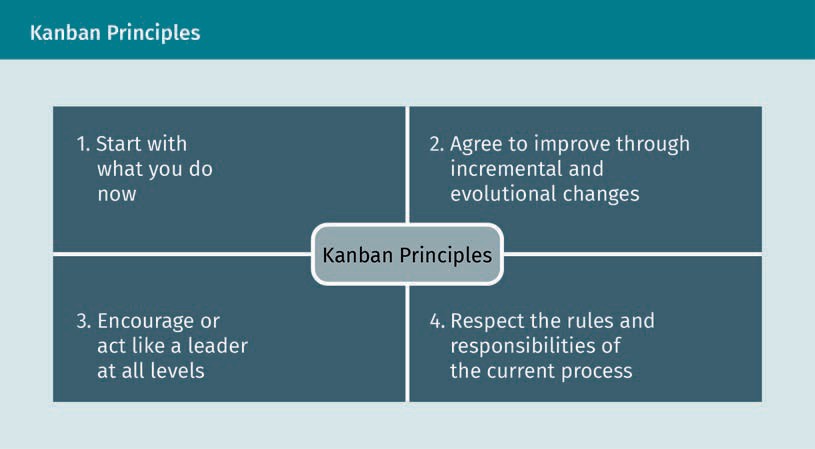
Agiles Projektmanagement



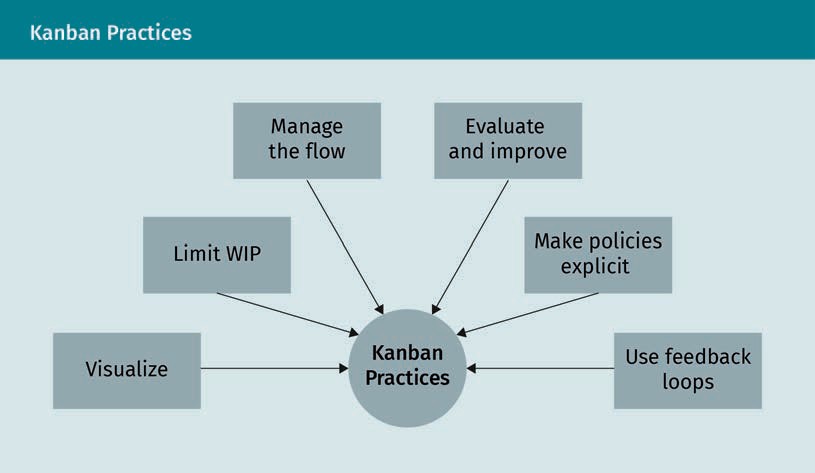
Aufgrund des technischen Fortschritts und des wachsenden Einsatzes des Internets bieten viele Online-Plattformen heute Kanban an. Dennoch folgen viele Firmen der traditionellen Kanban-Philosophie, bei der tatsächliche Tafeln bevorzugt werden, da sie den Austausch fördern (Collado, 2018). Durch das Kanban-Board 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



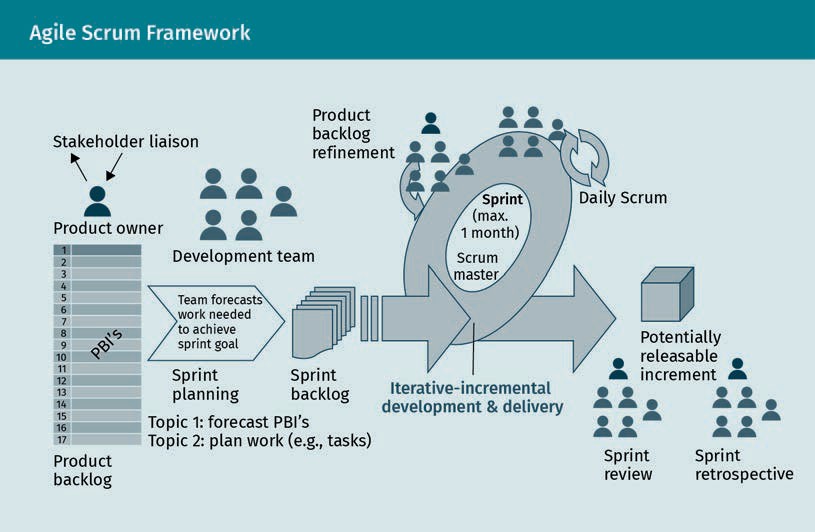
Die Methode des Agile Scrum wurde von Schwaber und Babatunde am DuPont Forschungszentrum und der Universität von Delaware in der Softwareentwicklung eingeführt. Sie wurde zunächst für die Teamführung verwendet und dann in der Easel Software Development Corporation angewandt (Sutherland, 2004). Entwicklungsteams 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.

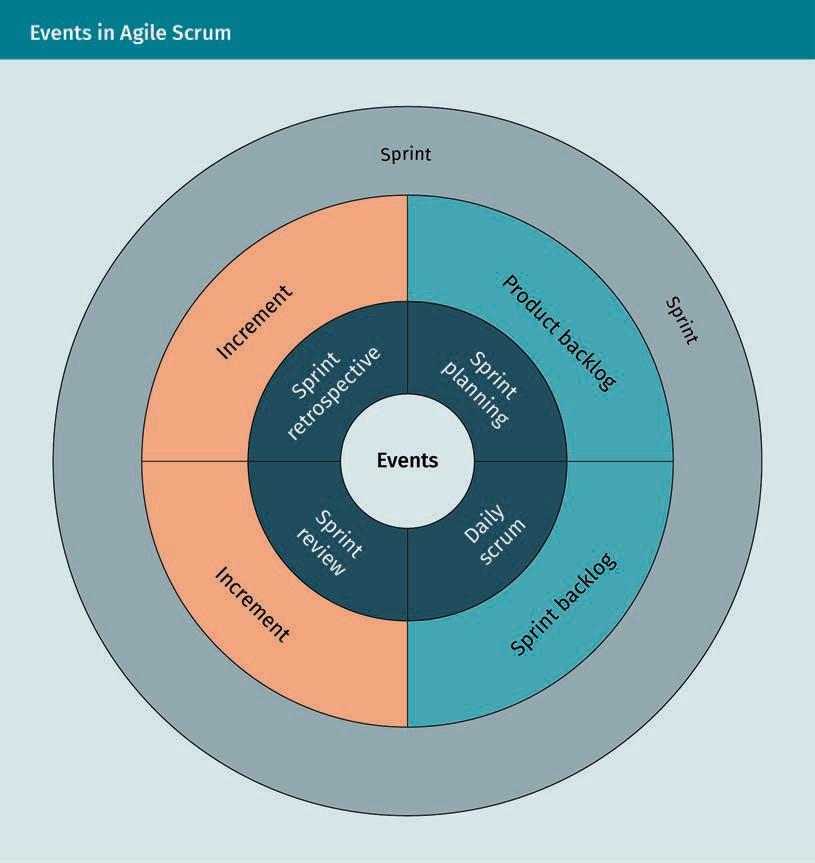
###### Artifacts als essenzielle Elemente des Agile Scrum

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 normallerweise 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

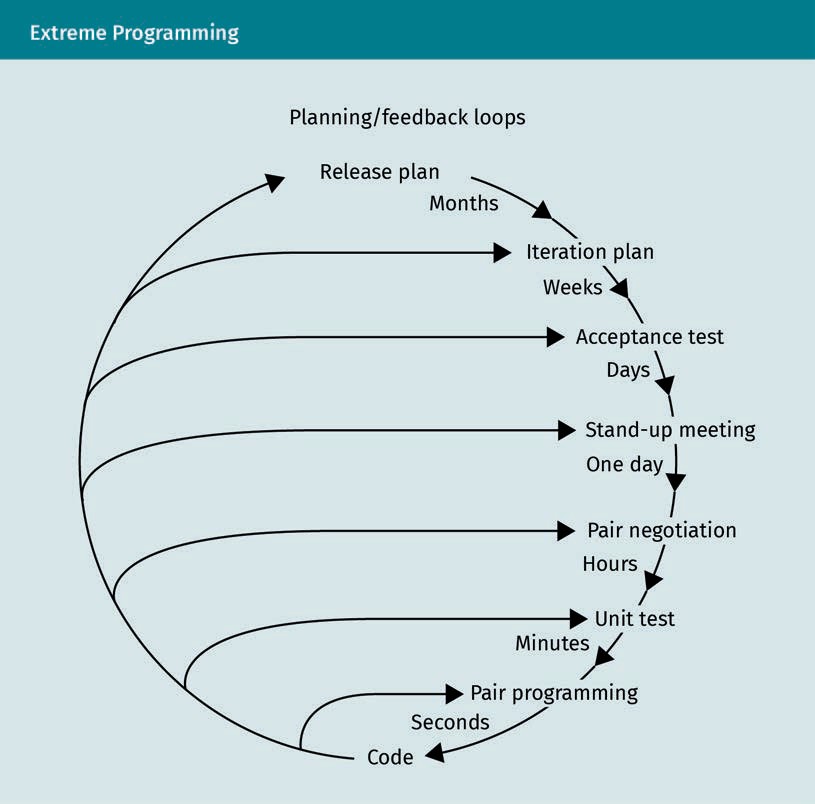
Agile Scrum eignet sich besonders für Branchen mit komplexen Projektprozessen, die Ergebnisse bewerten müssen und die um Kundenzufriedenheit bemüht sind (Jongerius et al., 2013). Am meisten hat die Softwarebranche und vor allen Dingen kleine Projektteams von maximal zehn Leuten davon profitiert. Agile 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 so schnell wie möglich stabile und hochwertige Software 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 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 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, anhand deren Benutzer produktiver sein können, erstellt werden.

###### 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 zueinander stehen und ihre Zusammenarbeit definitionsgemäß aufgrund der unterschiedlichen Ziele paradox ist. Das Entwicklungsteam versucht so schnell wie möglich Software zu erstellen und zu aktualisieren, während das IT-Team Änderungen 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.

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 Wartung, Updates, Außerbetriebnahme und Ersatz der Software. Durch die Kombination dieser Faktoren werden 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

Es wäre wünschenswert, dass Entwicklung und Betrieb 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 ihre Reaktion auf Änderungen zu beschleunigen. Es ist kein kleiner Schritt, der kulturelle Änderungen bei der Softwareentwicklung mit sich bringt und einen Baustein auf dem Weg zur Cloud darstellt. 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 einfache 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 für eine 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, Tools und Techniken, die bei der Verknüpfung von Vorgehensweisen helfen, so dass sie zu einem integralen Bestandteil des täglichen Arbeitsablaufs werden. Werkzeuge können auch zur Durchsetzung von Best Practices, z.B. zum Austausch von Problemlösungsdaten zwischen der Entwicklung und dem Betrieb, verwendet werden. Softwaretools werden beispielsweise zur Prüfung des Systems, nicht nur in der Entwicklungsumgebung, sondern auch in der Qualitätssicherung und in der Produktion eingesetzt. Es gibt Softwaretools, die während der Codeausführung zur Fehlerfindung, Timeoutmessung, Geräteparameteruntersuchung usw. nützlich sind. Die gespeicherten Protokolle können anhand von Monitoring-Tools im IT-Betrieb eingesehen werden. Dadurch können Codierungsprobleme leichter gefunden und Programmfehler 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 Diskrepanz 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 sie 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 Geschäft und Entwicklung zu vermindern, ist die Verwendung von agilen Prozessen. Obwohl es im Rahmen von DevOps keine direkte Lösung für dieses Problem gibt, ist bereits 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 für bessere Kommunikation ist ein Grundkonzept, 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 kurz erklärt:

Planen

Bevor Entwickler mit dem Codieren beginnen, muss zunächst ein gründlicher Plan des gesamten Workflows stehen. Dabei spielen Produkt- und Projektmanager eine wichtige Rolle. Es liegt in ihrer Verantwortung, einen Produktionsplan zur Führung des gesamten Teams durch den Prozess aufzustellen. Nach der Sammlung von Input und relevanter Informationen von den Benutzern und Stakeholdern wird der Auftrag auf einer Liste in einzelne Aufgaben 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. 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.

DevOps

Build

Die Build-Phase ist deshalb wichtig, da dabei Codierungsfehler zum Vorschein kommen, bevor sie durch die Pipeline geschickt werden und eventuell größere Probleme verursachen. Typischerweise wird durch den Pull Request ein automatisierter Prozess angestoßen, bei dem der Code in einen Build, ein bereitstellbares Paket oder eine ausführbare Datei kompiliert wird. Dies trifft natürlich nur auf Programmiersprachen zu, die kompiliert werden müssen. Zum Beispiel müssen 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 welche Tests zu diesem Zeitpunkt durchgeführt werden sollen. In dieser Phase ist das Testen günstig, denn ein neuer Test kann eingeführt werden, ohne den Entwicklerfluss zu unterbrechen oder die Produktionsumgebung zu stören.

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 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 dieser Phase kann ein manueller Genehmigungsmechanismus eingeführt werden, durch den nur bestimmte Mitarbeiter einen Release in die Entwicklung genehmigen können. Die Technologie existiert. 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, wobei 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. Auch das bietet das Unternehmen als Dienst an, genauso wie die passenden Feedbacktools, 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 generierten Kundenfeedback auf. 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 während dessen Lebensdauer. Diese 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 zu Installierung, Verwaltung und 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 Produktionsumgebungen 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 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 einem gemeinsamen Entwicklungszweig bzw. einem 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).

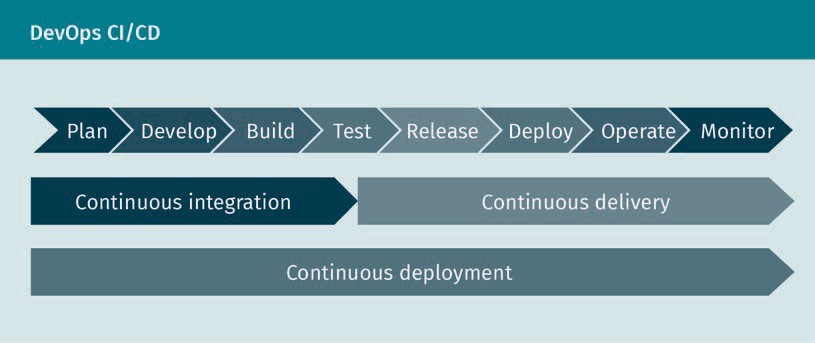
Kontinuierliche Lieferung

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. Alle 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 Fehlern 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 Untersuchung 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 DevOps-Organisationen bevorzugt werden.

Silos

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

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 zu liefern. 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. Als Misserfolgsindikatoren gelten

* + - Völlig fehlende Änderungsbereitschaft,
    - Schwache Änderungsbereitschaft und
    - Niedrige Einsatzbereitschaft der Mitarbeiter.

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.

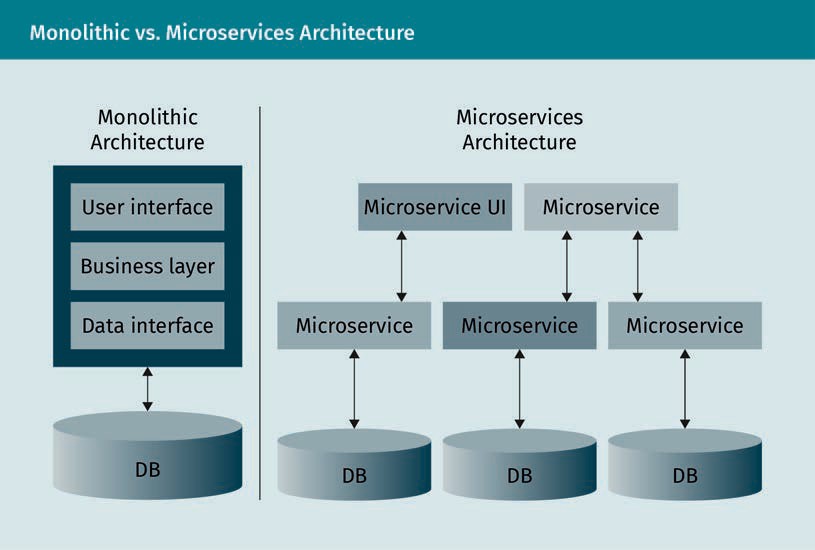
###### Die richtige Mischung

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, durch die Firmenkulturen heute mitbestimmt werden. 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 des 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 der Implementierung von DevOps, eine bedeutende Rolle zu. Ein weiterer Vorteil des Microservice-Ansatzes ist es, dass kleinere Teams einzelne Dienste schneller erstellen, validieren und freigeben können, wodurch die Einführung von DevOps ebenfalls vereinfacht wird. Microservices und DevOps sind also sehr komplementär.

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

###### Twelve-Factor-App

Die Twelve-Factor-App, im Deutschen 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 im Rahmen von DevOps voll mit Microservices 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 Kommunikations-endpunkt in einem Computernetz, der Anwendungen über das Internet bei der Kommunikation miteinander 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-Tools 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 in dieser Lerneinheit 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 Systemlebens-zyklus abdecken.

Ordnungsgemäßes Herunterfahren

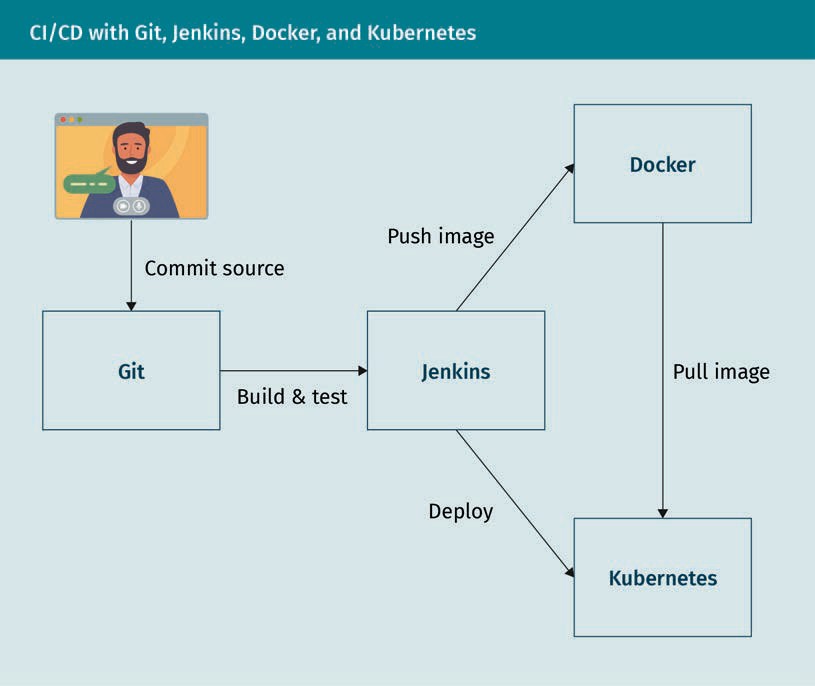
Die Fähigkeit eines Betriebssystems, alle Prozesse sicher abzuschalten und Verbindungen sicher abzuschließen, wenn im laufenden System ein Abschaltesignal eingeht.

Container

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

Versionsverwaltungssystem

Ein 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

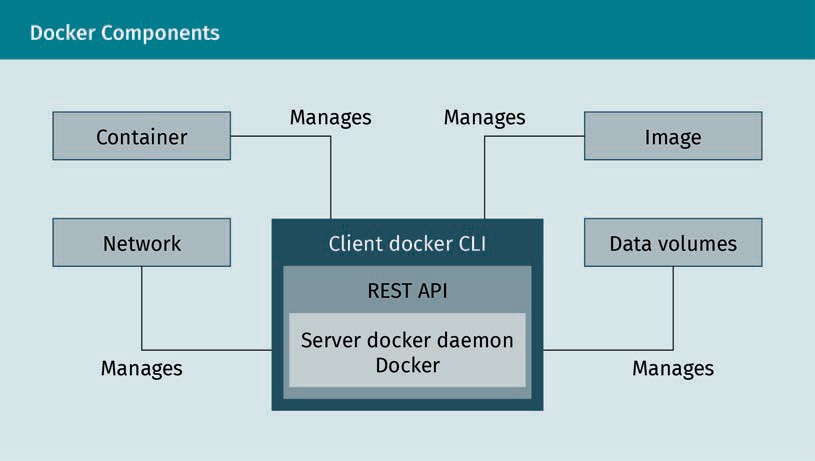
###### Docker

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 Images.
  + Image: Ein Docker-Image besteht aus Elementen, wie Code, Config-Dateien, Umgebungsvariablen, Bibliotheken und Runtime-System, die zur Ausführung einer Anwendung als Container notwendig sind.
  + Container: Eine Standardeinheit, die spontan erstellt werden kann, um eine bestimmte Anwendung oder eine Umgebung bereitzustellen.
  + Docker-Registrierung: Ein Dienst für Docker-Images und -Repositories.
  + Docker-Daemon: Ein Server, hier ein langes Serverprogramm, das als Daemon-Prozess bezeichnet wird. Ein Daemon ist ein Linux- oder UNIX-Programm, das im Hintergrund abläuft und daher auch als Hintergrundprogramm bezeichnet wird.
  + Docker-REST-API: Schnittstellen, die Programme zur Kommunikation mit dem Daemon und zur Übermittlung von Anweisungen verwenden können.
  + Docker-Client: Der Client hat eine Kommandozeile, 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.

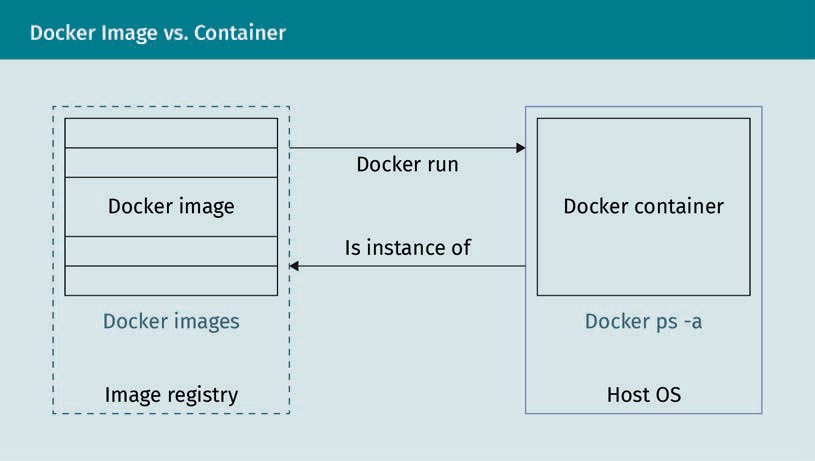


Ein Container wird durch ein Image gestartet (Container bestehen aus Images). Ein Image ist ein auszuführendes Paket, das alles, was zur Ausführung nötig ist enthält, nämlich

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

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-Images notwendig ist.

* + Das Argument python:3 ist der Name und das Tag für das grundlegende Image, wobei das Tag nach „:” kommt. Der Name des Images ist also python und das Tag ist 3.
  + Durch ADD wird das Skript dem Image zugefügt.
  + Und CMDlöst 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 .

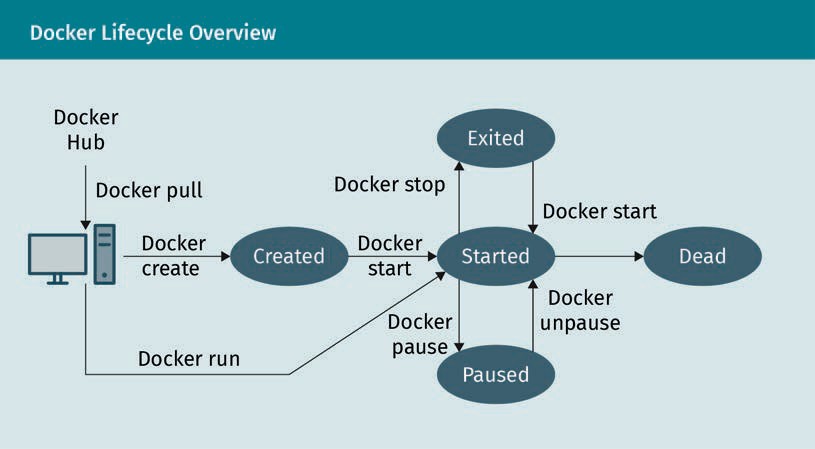
Nach der Image-Erstellung kann ein Container mit folgendem Befehl ausgeführt werden:

$ docker run my-image

In obigem Befehl steht my-image für den Namen des Images, das zuvor erstellt wurd 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 da ist, 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 die 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 mit erheblichem Zeitaufwand und ohne ausreichende Qualitätskontrolle manuell ausgeführt werden müssten (Red Hat Ansible, o. D.-b). Das folgende Diagramm zeigt einen CI/CD-Setup, bei dem Ansible zur kontinuierlichen Software-Verteilung verwendet wird.

DevOps



Ansible ist Software für die zentrale Verwaltung von verteilten Servern. Die Community-Version von Ansible ist lizenzfrei 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, wie 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 Lernaufwand 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 werden sollen, müssen über ein Netzwerk erreichbar sein. Dabei spielt es keine Rolle, ob dies über das Internet, eine LAN- oder eine VPN-Verbindung geschieht.
    - SSH-Schlüssel: Die Kommunikation zwischen Ansible und den entfernt gelegenen Hosts findet hauptsächlich über SSH statt. Für den passwortlosen Zugriff von Ansible auf die entfernten Server vom zentralen Host aus, 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 mehrfach wiederhol- und einsetzbares Framework für die Bereitstellung auf mehreren Computern bieten. 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 Cloud-Umgebung dynamisch geändert werden. Dadurch hat der Kunde 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 Wachstumspotenzial hat. Skalierbarkeit bedeutet, dass ein Unternehmen Systeme so konfigurieren kann, dass sie bei hohem Bedarf ausbaufähig sind, und wenn der Bedarf zurückgeht, zurückgefahren werden können. Einige für DevOps typische Aktivitäten bieten sich zur Erreichung optimaler Skalierbarkeit an. Dazu gehören Kommunikationsmöglichkeiten, Ausrichtung auf Hochleistung, mehr Möglichkeiten für Kreativität und schnellerer Anwendungs-Release. Container-Orchestrierung ist bei Höchstauslastung ein Muss. Durch Container-Orchestrierung werden Provisioning, Verwaltung, Skalierung und Networking von Containern automatisiert. Unternehmen, die hunderttausende von Containern und Hosts bereitstellen und verwalten müssen, kommt die Container-Orchestrierung besonders zugute (Red Hat Ansible, o.D.-c). Container-Orchestrierung lässt sich in jeder Umgebung, in der Container eingesetzt werden, verwenden. Mit ihrer Hilfe lassen sich Anwendungen in verschiedenen Umgebungen ohne besondere Anpassung bereitstellen. Mit Microservices in Containern können Dienste, wie Speicherung, Networking und Sicherheit, leichter arrangiert werden (VMware, o.D.).

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

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

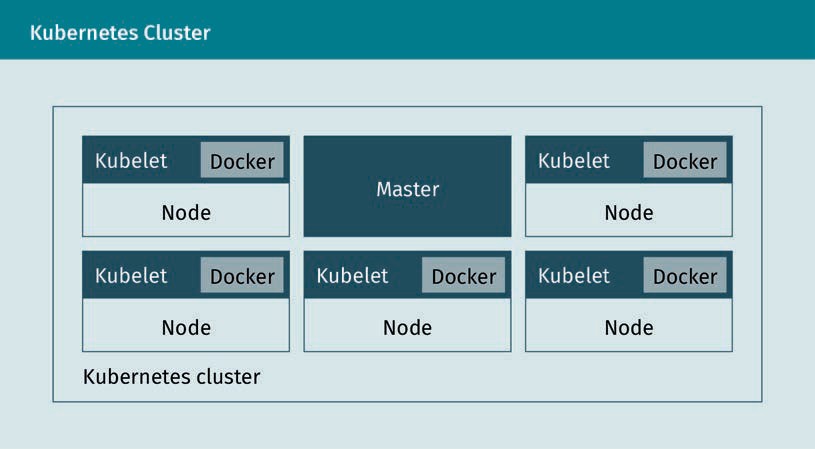
Tools zur Container-Orchestrierung bieten ein Framework für die groß angelegte Verwaltung von Containern und Microservicearchitekturen. Es gibt viele Container-Orchestrierungs-Tools, die zum Container-Lifecycle Management verwendet werden können. Zu den am häufigsten eingesetzten gehören Kubernetes,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

###### Kubernetes

Kubernetes ist ein ursprünglich von Google entworfenes und entwickeltes, Open-Source-Tool für die Orchestrierung für Containern. Mit einer Kubernetes-Orchestrierung können Anwendungsdienste über mehrere Container hinweg entwickelt werden, Cluster-übergreifend Containerplanung und -skalierung vorgenommen und ihre Integrität auf Dauer überwacht werden (Kubernetes, o.D.). Viele der mit der Bereitstellung und Skalierung von Containeranwendungen verbundenen, manuellen Prozesse fallen mit Kubernetes weg. Container können auf Cluster-Host-Gruppen aus physischen oder virtuellen Maschinen ausgeführt werden, da Kubernetes eine gute Plattform für die einfache und effiziente Verwaltung dieser Cluster bietet.

Cloud Native Ein Ansatz, bei dem gewährleistet wird, dass sich Design und Entwicklung einer 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 Cloud-Anwendungen, für die schnelles Skalieren notwendig ist. Workloads sind mit Kubernetes auch portierbar und Lastenausgleich ist möglich, da Anwendungen ohne Neuentwicklung übertragen werden können. Das folgende Diagramm zeigt einen Kubernetes-Cluster:



Ein Kubernetes-Cluster besteht aus zwei Ressourcenarten:

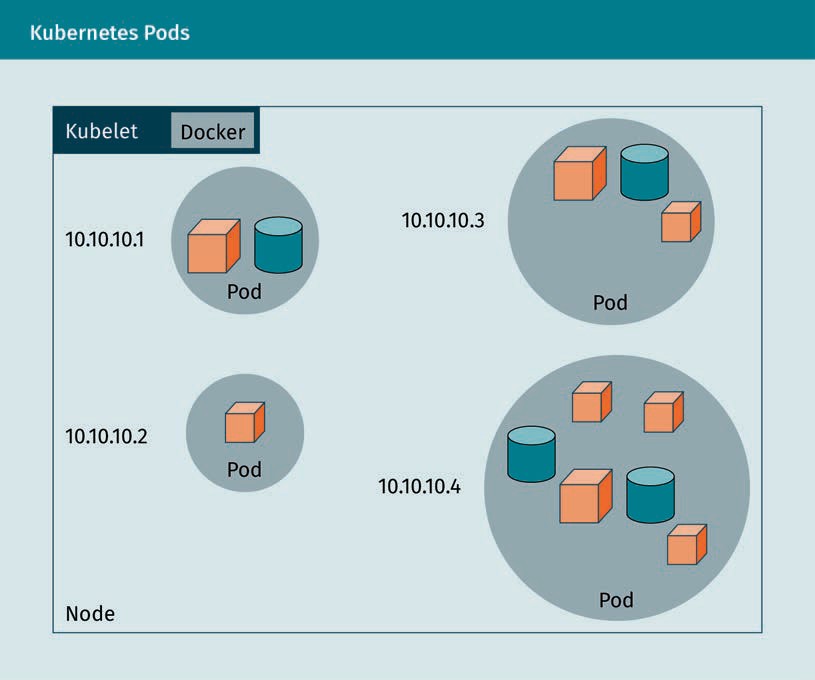
* Master, d.h. einem Clusterkoordinator
* Knoten, d.h. den Workern, die die Anwendungen ausführen

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 Abwicklung des Containerbetriebs, wie z.B. Docker, haben. Ein für die Produktionslast verantwortlicher Kubernetes-Cluster sollte mindestens drei Knoten haben. Wenn Anwendungen auf Kubernetes bereitgestellt werden, muss dem Master der Befehl zum Starten der Anwendung gegeben werden. Der Master plant die Ausführung der Container auf den Cluster-Knoten. 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 mehreren Containern 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.

DevOps

Ein Pod ist ein Modell für einen anwendungsspezifischen Logik-Host, der einen oder mehrere, relative eng verbundene Anwendungscontainer enthält. In Umgebungen außerhalb der Cloud gleichen Programme auf derselben physischen oder virtuellen Maschine den Cloud-Anwendungen auf demselben Logik-Host.

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



###### 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 sowie 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 Festlegung 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 Container-Image 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 Aufgabe 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, Release-Verwaltung, 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 sind Sie in der Lage...

… different testing paradigms in the software development lifecycle.

… different approaches to testing while developing, such as test-driven development (TDD) and behavior-driven development (BDD).

… the terms continuous integration (CI), continuous delivery (CD), continuous testing (CTe), and continuous training (CT).

… how to apply the CI/CD approach to improve machine learning systems.

… how to track changes in a software development project using tools.

… what integrated development tools (IDEs) are and some example of development tools.

DL-E-DLMDSSEDIS01-U03

1. Softwareentwicklung

### Einführung

After ﬁnishing with the development of a machine learning (ML) solution, the project team does not deploy the software immediately to the production environment to serve customers. There is another step between the development and deployment: testing. The testing procedure is a cross-functional and continuous activity in which the whole team is involved. In an ideal case, the testing team will collaborate with the developers and users to develop and improve the automated testing at the very begin- ning phases of the software project. These automated tests will be developed by the testing team before development of the modules by the development team. If the developed modules pass the relevant tests, it demonstrates the successful implemen- tation of the required functionalities in the software. The process of identifying and pri- oritizing the project’s risks and designing measures to mitigate those risks is called testing strategy. A well-designed testing strategy ensures working software functionality (meaning fewer bugs and lower support costs), and deﬁnes a framework for develop- ment practices. Different testing types from the perspective of different stakeholders of a project will be discussed at the beginning of the unit, after which we will discuss how to test and develop in parallel. The scenario for a machine learning solution project as a non-deterministic system is more complex and requires a revision of the conven- tional testing methods. At this point, we will also learn how to automate the develop- ment pipeline. We will then discuss automated development, in which it is even more crucial to keep track of changes.

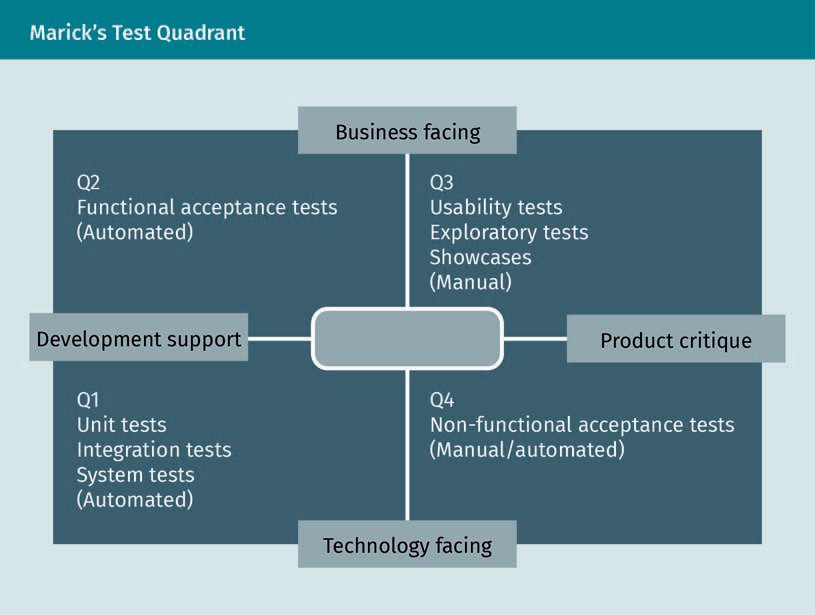
### Testparadigmen und Überwachung

To deliver a high-quality software solution, Brian Marick introduced several test scenar- ios shown in the ﬁgure below (known as Marick’s quadrant), which the project team should implement and perform before releasing the software solution (Humble & Far- ley, 2015). These tests are categorized into two dimensions:

1. Tests that support development (help the developers to build the software with conﬁdence) and tests that critique the product (help to discover insufﬁciencies and deﬁciencies in the software programs)
2. Tests that are business-facing (developed from the perspective of a business per- son) and tests that are development-facing (developed from the perspective of a developer)

In the following, we will discuss these four scenarios in more detail.

Softwareentwicklung



Quadrant 1 (Q1)

Among the technology-facing tests that support the development team there are unit tests, integration tests, and system tests. These tests are normally written by the devel- opment team. In a unit test, a unit of the software will be tested in an isolated environ- ment without interaction with the other components (units) of the system, i.e., to per- form a unit test, interactions with other units of the system will be simulated (to provide the required inputs). This isolation results in the fast execution of unit testing. However, due to this isolation, unit testing cannot ensure the functionality of the unit as it interacts with other software components. Therefore, we need another suite of tests, namely integration tests. In an integration test, a wide range of functionalities embedded in different units of the software product will be tested. In such a test, not only the units’ functionalities but also the interactions and the interfaces between the units will be tested. Through integration testing, it is ensured that each independent unit of the software solution works seamlessly with the services it depends on. An integration test of the system under test (SUT) could be executed against the real external systems it depends on or against a test harness as a part of the software product. Sys- tem testing is the testing of the software product as a whole to assess its compatibility with the client’s requirements speciﬁed in the software requirement speciﬁcation (SRS) document. In integration testing, we perform a functional test on a collection of the units and their interaction; in system testing, we test the system as a whole. System testing follows integration testing. Before deploying a software solution into the pro- duction environment, a deployment test is crucial. Such a test will check if, for example, the software solution is installed and conﬁgured correctly and is able to communicate with the required services.

Test harness In software quality assurance, this is a collection of soft- ware and test data that is used for automated testing of a software under dif- ferent environmental conditions as well as

monitoring its behavior and out-

puts.

Functional test The requirements that specify what a software system must or must not do are called functional requirements. Non- functional require- ments specify “how” the system should

do things.

Test engineer In a software project, the person or team responsible for deﬁning, executing, and reporting the test procedure dur- ing the software development lifecy- cle is called the test engineer (or team).

Quadrant 2 (Q2)

Tests that support the development team on the business-facing side are commonly known as functional acceptance tests. An acceptance test ensures the operation of some pre-deﬁned criteria such as functionality, capacity, usability, and security. An acceptance test that concerns a functionality criterion is called a functional acceptance test. When such a test is passed, the development team and users can be sure that the functionality of the developed unit is implemented correctly. End-to-end testing (E2E testing), when the developers test if the whole software product from beginning to end behaves as expected, is also an example of this category (Q2). In writing this test, the system’s dependencies are deﬁned, and the test ensures that all pieces of the software solution work together according to the speciﬁcations. The main goal of E2E testing is to test the software product from the viewpoint of the user by simulating real user sce- narios (Katalon, 2020). In system testing, only the speciﬁc software system is tested; in E2E testing, the speciﬁc software system is tested together with the connected external systems.

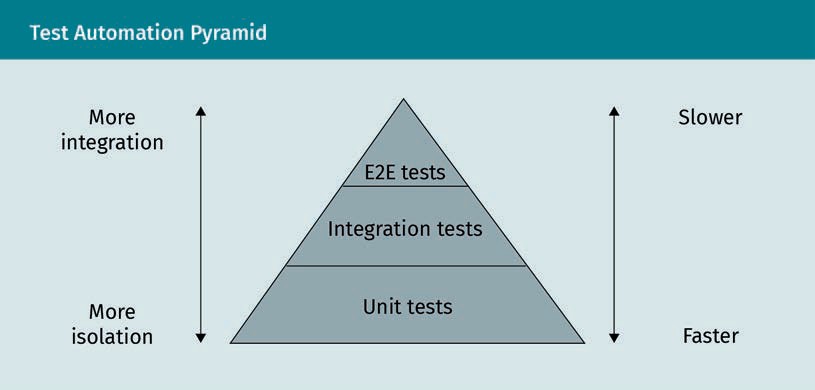
Quadrant 3 (Q3)

Business-facing tests that critique the project are mostly manual tests that verify if the software product delivers the expected values to the customer. It checks not only the speciﬁcations but also whether the speciﬁcations are deﬁned correctly. Some examples of tests in this category are showcases, exploratory testing, and usability testing. At the end of each iteration, the development team performs showcases to demonstrate the new functionalities to users. This demonstration of functionality during the develop- ment phase prevents any misunderstanding or speciﬁcation problem. Another example of testing, exploratory testing, is deﬁned by J. Bach (2003) as manual testing where “the test engineer actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests” (p. 2). Exploratory testing results in new sets of automated testing. Finally, usability testing is performed to test how the users could easily accomplish the deﬁned goals using the developed software.

Quadrant 4 (Q4)

Technology-facing tests that critique the project verify the nonfunctional criteria of the software system like capacity, availability, and security. Nonfunctional acceptance test- ing is designed for this purpose. This type of testing could also be fully automated, but it is performed less frequently than functional acceptance testing and usually at the end of the development pipeline. After getting familiar with different types of software testing, now the question is how to design an automated testing suite. An effective automated testing strategy could be divided into three levels, structured by the so- called test automation pyramid as shown in the ﬁgure below (Cohn, 2009). In this model, the unit test forms the base of this test automation pyramid. The next level is integration testing, and at the top of the pyramid is the end-to-end testing (E2E). The testing pyramid is the original representation of this approach and it could be exten- ded to contain more automated test methods (mostly from the Q1 and Q2 regions in Marick’s test quadrant).

Softwareentwicklung



###### Machine Learning Models Testing

Unlike traditional software products, machine learning (ML) models are non-determin- istic, but there are several possibilities for the transition to the next state for the same input. As in such a software solution, the responses of the system are adapted to what the system has learned in previous transactions. Therefore, in machine learning solu- tions, the tester should test data, code, and the learning program, as well as the frame- works (e.g., TensorFlow) that support the ML development. In addition, developing test oracles is time- and labor-consuming, as domain-speciﬁc knowledge is required (Zhang et al., 2020). An oracle is a mechanism used in software testing and software engineer- ing to determine whether a test was successful (Kaner, 2004). It is used by comparing the output of a system under test, given as input to a speciﬁc test case, with the result that the product should provide (determined by the oracle).

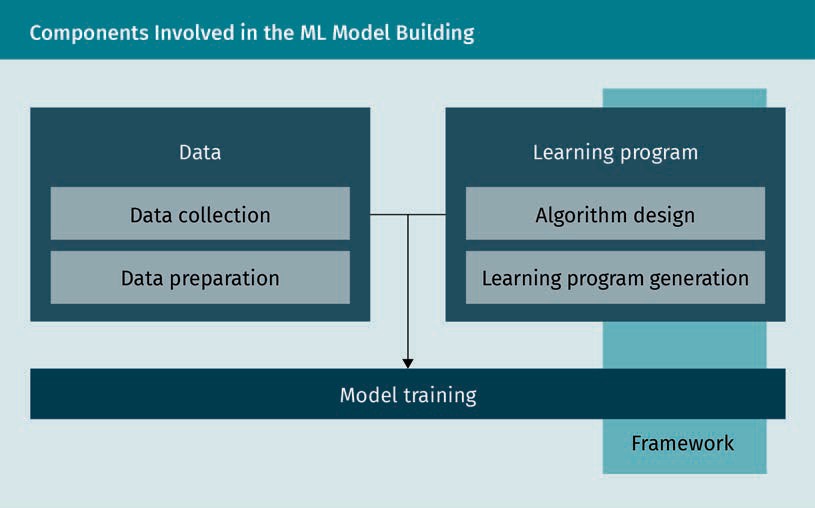
The components involved in a machine learning model building are represented in the ﬁgure below (Zhang et al., 2020). To test an ML software solution, testers should test all these components, as well as their interactions.

Non-determinism This is a concept from theoretical computer science in which algorithms or machines cannot only perform exactly one calculation for a certain input (deter- ministic). There are instead several pos- sibilities for the transition to the next state for the same input.

TensorFlow This is an end-to- end open-source platform for machine learning that includes a com- prehensive, ﬂexible ecosystem of tools, libraries, and com- munity resources. It lets researchers push the state-of- the-art in ML and developers easily build and deploy ML

-powered applica- tions (Tensorﬂow,

n.d.).



Testing data

In testing data components of an ML model, test engineers should check issues such as whether the data are sufﬁcient for the training; if the data fairly represent future data; whether data contain a lot of noises (such as biased data); and if there is a skew between the training and test data (Zhang et al., 2020). Although it is mostly the task of the ML developer to design the right test cases, there are some general hints (Heck, 2020).

* Data type and schema: ML engineers always check the format, type, and schema of data to ensure data quality and integrity. This test helps the ML engineers to ensure that data are clean from malformatted inputs (Kim et al., 2018).
* Implicit constraints: ML engineers often examine certain constraints, like assertions in software testing. Such constraints are not regarding single data points, but rather how the subgroups of data relate to other subgroups (Kim et al., 2018). For example, the number of downloaded software licenses from a speciﬁc region should be equal to the number of purchased licenses.
* Testing platforms: There are some ML testing frameworks from companies like Goo- gle’s TFX (Breck et al., 2019) and Apple Overton (Ré et al., 2019). For instance, Google provides a data validation system that lets developers deﬁne a data schema to check data properties and to generate synthetic data to test the model (Breck et al., 2019).

Testing the learning program

A learning program (ML model) is composed of the algorithm and the code to imple- ment and to conﬁgure it. A trained ML model should be tested according to metrics such as correctness, fairness, and interpretability (Zhang et al., 2020). Correctness includes accuracy, precision, and recall. To explain the difference between precision and recall, consider an ML algorithm that classiﬁed 100 tumors as malignant (the posi- tive class) or benign (the negative class): true positives (TPs), false positives (FPs), false

Softwareentwicklung

negatives (FNs), and true negatives (TNs) (Google, 2020b). Precision is deﬁned as the proportion of actually correct positive identiﬁcations (TP/(TP+FP)). A recall is the pro- portion of the actual positive cases that have been identiﬁed correctly by our algorithm (TP/(TP+FN)). A fair ML model makes fair predictions across various demographic groups (Dwork, 2012). IBM has developed a tool for ML fairness called AI Fairness 360 (IBM, n.d.-a). This tool assists the ML developer to examine, report, and mitigate dis- crimination and bias in ML models throughout the AI application lifecycle. Finally, an ML model is assumed as an interpretable model when an observer/user can under- stand the cause of the decision produced by it. There are some tools for checking inter- pretability such as the Machine Learning Interpretability module from H2O (H2O, n.d.).

###### ML Model Monitoring

The ML system should be monitored to detect any unexpected behavior due to changes in the input data. Monitoring is essential for models that automatically digest new data in a real-time fashion during the training, therefore, it is always required that these models also predict in real-time. Four classes of monitoring ML systems could be deﬁned as follows (Gade, 2019):

1. Feature monitoring, to guarantee that model features are stable over time, data invariants are stable, to have a continuous overview of statistics
2. Model operations monitoring, to detect staleness, regressions in serving latency, throughput, RAM usage, etc.
3. Model performance monitoring, to detect regressions in prediction quality
4. Model bias monitoring, for detection of unknown bias

Amazon SageMaker Model Monitor is an example of an ML monitoring tool that allows developers to detect and remediate concept drift (Amazon, n.d.-b). SageMaker Model Monitor automatically detects concept drift in deployed ML models and provides detailed alerts that help identify the source of the problem.

### Approaches to Development and Testing

In software engineering, a software development methodology (also known as lifecycle) applies to the software development project and breaks it into separate phases or stages. Each phase includes activities with the purpose of more efﬁcient planning and management of the project. The waterfall model is one of the traditional methodolo- gies, while the agile models, like Kanban and Scrum, are the more modern approaches (Lumen, 2020). In this section, we brieﬂy discuss three software development techni- ques that support agile development methodologies by enhancing the testing practice and assisting automated testing:

* + Test-driven development (TDD)
  + Behavior-driven development (BDD)
  + Acceptance test-driven development (ATDD)

Concept drift

If the data being used to generate predictions differs from data used to train the model, the patterns the model uses to make predic- tions no longer apply and this is called concept drift.

Agile model In this approach, speciﬁcations, requirements, and code change dynam- ically through collab- oration between cross-functional

teams.

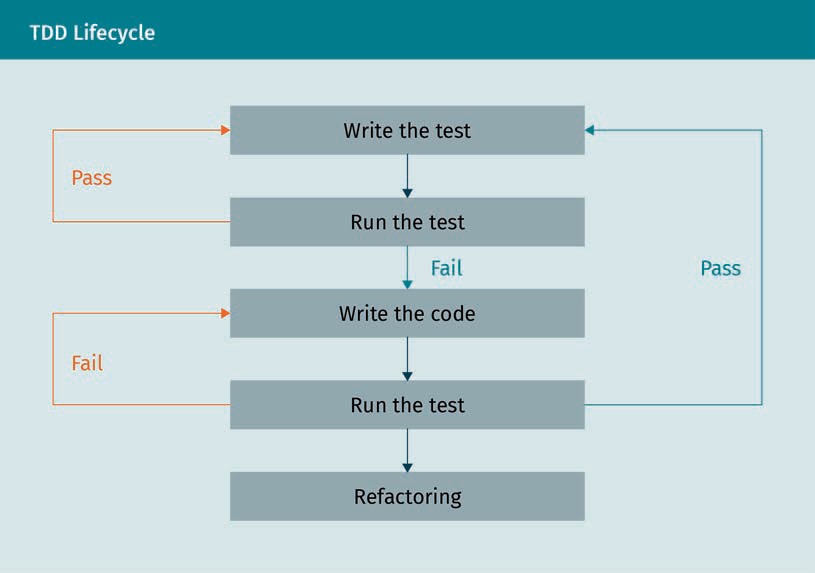
Refactoring In the software development lifecy- cle, the process of cleaning and simpli- fying the design of an existing code, without changing its behavior and func- tionality, is called

refactoring.

###### Test-Driven Development (TDD)

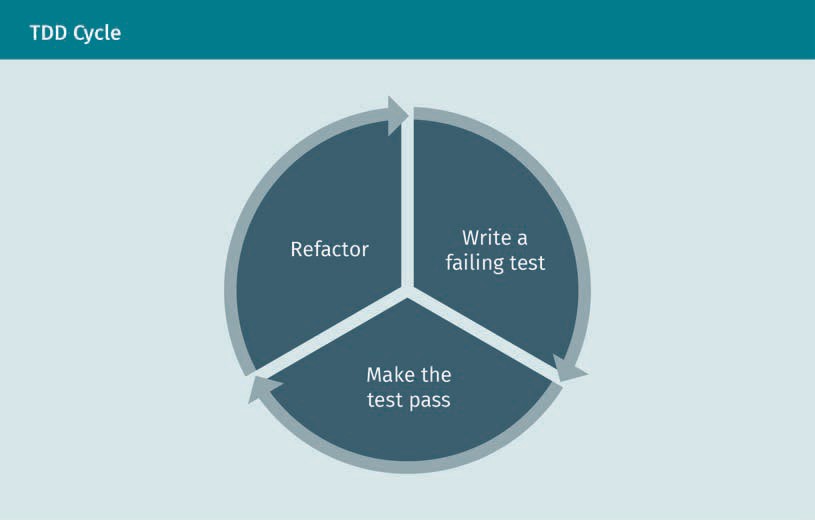
Test-driven development is a testing methodology performed from a developer’s per- spective. In this method, a quality assurance engineer begins designing and writing test cases for every functionality of the software product. This technique tries to answer a simple question—is the code valid? The main purpose of this technique is to revise or write fresh code only when the test fails, resulting in less duplication of test scripts. This technique is most common in agile development. In a TDD approach, automated test scripts are written before functional pieces of code. The TDD lifecycle can be split into the following steps (Beck, 2014):

1. Write the test. This test should deﬁne (or improve) some speciﬁc functionality. For this reason, the test developer should be familiar with the product’s requirements. This is what differentiates TDD from traditional development approaches, i.e., ﬁrst writing the test and then the code.
2. Run the test and verify that it fails, as the functionality has not been written yet.
3. Write the code for the functionality that should pass the written test. In this step, the developer should write the simplest code that will make the test work.
4. Test the new code with the written test. It should pass the test.
5. Refactor to remove duplicate code and clean up the code during the TDD process. In this step, all naming conventions should also be checked and corrected.

The aforementioned steps are illustrated in the following ﬂowchart:

Softwareentwicklung

In summary, the TDD follows the pattern of write a failing test, make the test pass, refactor, and continue in this loop (Freeman & Pryce, 2012).



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

By using test-driven development (TDD) in projects in different environments, some confusion and misunderstanding may arise; programmers may not know where to start; what to test and what not to test; how much to test in one go; what to call their tests; and how to understand why a test fails (North, 2020). One approach to reduce such confusions is behavior-driven development (BDD), which is an agile software develop- ment practice introduced by Dan North (2006). BDD aims to provide a shared under- standing of how an application should behave by discovering new features based on concrete examples. Those key examples are formulated using the natural language in a “given/when/then” structure.

* Given: Some initial context (the givens)
* When: An event occurs
* Then: Some outcome is ensured

An implementation of the “given/when/then” approach could resemble the following.

* + Given: The user has entered valid login credentials.
  + When: A user clicks on the login button.
  + Then: The successful validation message is displayed.

Using such simple plain language (English here) helps all stakeholders (developers and clients) of the project to understand the feature behavior of a functionality. Gherkin is a domain-speciﬁc language for describing examples with “given/when/then” in plain text ﬁles, called feature ﬁles. Feature ﬁles describe how a system should interact with the user. A simple example of a feature ﬁle using Gherkin looks like

Feature: Calculator

Scenario: “+” should add to current total Given the current total is “5”

When I enter “7”

Then the current total should be “12”

The scenario in the example above describes a step in a “given-when-then” structure (Specﬂow, n.d.).

###### Acceptance Test-Driven Development (ATDD)

In the acceptance test-driven development (ATDD) method, a single test is developed from the perspective of the client with the focus on accurate functionality. Indeed, ATDD and BDD are very similar and the main difference is in their focus: ATDD focuses on the functionality and BDD focuses more on behavior. In this development method, different project stakeholders with different perspectives are involved in writing accept- ance tests before implementing the relevant functionality. These acceptance tests will reﬂect the user’s perspective. The key differences between TDD, BDD, and ATDD are explained in the following table (Unadkat, 2020).

|  |  |  |  |
| --- | --- | --- | --- |
| TDD vs. BDD vs. ATDD | | | |
|  | TDD | BDD | ATDD |
| Deﬁnition | Is a development approach to imple- ment a feature | Is a development approach based on the system behav- ior | Is a development approach for cap- turing the require- ments (similar to BDD) by writing acceptance tests before implement- ing the relevant functionality |

Softwareentwicklung

|  |  |  |  |
| --- | --- | --- | --- |
|  | TDD | BDD | ATDD |
| Main Focus | Unit tests | Understanding requirements based on system behavior | Writing acceptance tests |
| Participants | Entwickler | Developers, cus- tomers, qual-  ity assurance engi- neers | Developers, cus- tomers, qual-  ity assurance engi- neers |
| Language | Similar to the main code | Plain English | Plain English |

We conclude this section with a brief discussion about pair programming, which is an agile software development technique in which two programmers work at one working station. In pair programming, one of the programmers, known as the driver, will write the code and the other developer, called the navigator, controls the code and concen- trates on the plan of action. It has been found that for a development-time cost of about 15 percent, pair programming enhances design quality, technical skills, and team communications while also reducing defects and stafﬁng risk (Cockburn & Williams, 2001). Pair programming was ﬁrst introduced as a part of the extreme programming software development method (Beck, 1999). Extreme programming is a set of software techniques that focuses on exchanging information, clarity, response, determination, and consideration. The goal of extreme programming is to have a piece of ﬁrm pro- gramming knowledge (Unadkat, 2020).

### Continuous Integration and Continuous Delivery

In this section, we focus on procedures for implementation and automation of continu- ous integration (CI), continuous delivery (CD), and continuous training (CT) of ML mod- els. To achieve this goal, the MLOps principles should be applied to the ML project. MLOps is a machine learning development approach that aims to link ML system devel- opment (ML) and ML system operation (Ops). Applying MLOps requires a focus on auto- mation and monitoring at all phases of ML system design, including integration, testing, release, deployment, and infrastructure management (Google, 2020b). We start this sec- tion with an introduction to continuous integration, continuous delivery, and continu- ous testing.

###### Continuous Integration (CI)

Continuous integration (CI) is a development philosophy in which developers commit their code to a version control repository periodically (normally teams have a minimal standard of committing code at least daily). The motive behind CI is to have a more straightforward method to recognize errors and other software quality problems on a smaller piece of code rather than larger chunks of code developed over a long period of time. Continuous integration was ﬁrst introduced by Kent (Beck, 1999). As with other extreme programming practices, the intention behind continuous integration was that, if regular integration of the codebase is good, why not do it all the time? Before imple- menting CI into the software project, the following components are required (Humble & Farley, 2015).

* + Version control: All elements in the software project should be checked into a ver- sion control repository like Git (Git, n.d.). This includes the source code, test scripts, database scripts, build and deployment scripts, conﬁguration ﬁles, etc.
  + An automated build: The build of the program should be able to run in an automa- ted way via the command line. It can start with a command-line script to command the IDE (integrated development environment) to build the software. However, it could also be a complex combination of multistage build scripts that call one another.
  + Agreement of the team: Continuous integration is teamwork (not a tool) and there- fore it requires commitment and discipline from the members of the development team. Only by the adoption of the required disciplines in the team can continuous integration result in an improvement in product quality.

There are several CI tools such as GoCD (GoCD, n.d.), CruiseControl (CruiseControl, n.d.), and Jenkins (Jenkins, n.d.-a). After installing the CI tool, one can start the CI process by conﬁguring the tool. The conﬁguration will specify the location of the source control repository, the required scripts to run to compile the software, and the automated commit tests.

###### Continuous Testing (CTe)

As we discussed in the ﬁrst section of this unit, automated testing assists the test engi- neers to write and run different types of the test during the software development life- cycle. The testing types range from a unit test to the system and regression test for the entire software (Sakolick, 2020). Regression testing is a type of testing in which you re- run the functional and non-functional tests on an already tested software product to check if the (modiﬁed) software passes the test after applying some new modiﬁcations (Basu, 2015). Regression tests follow other automated tests, such as performance tests, API tests, and security tests. All these tests are required to trigger through a command line or other automation tools. After automating the testing procedure, continuous testing (CTe) indicates that the automated test is already integrated into the CI/CD pipeline. In this case, some of the tests (like the unit and functionality tests) should have been integrated into the CI part (when CI identiﬁes the issues before or during the

Softwareentwicklung

integration process). Tests that need complete delivery conditions, such as perform- ance and security tests, should usually be integrated into the CD. These tests are exe- cuted after the build process (Sakolick, 2020).

###### Continuous Delivery (CD)

Continuous delivery (CD) is a software engineering method in which project teams con- tinually deliver valuable software in short cycles and guarantee that the software can be reliably delivered at any time (Chen, 2015). CD empowers companies to rapidly, efﬁ- ciently, and reliably deliver service improvements to the market. During the develop- ment process, each development team uses one or more development and testing environments to stage the application modiﬁcations for testing. This process can be automated using the CI/CD tools such as Jenkins (Jenkins, n.d.-c), CircleCI (CircleCI, n.d.), Travis CI (Travis CI, n.d.).

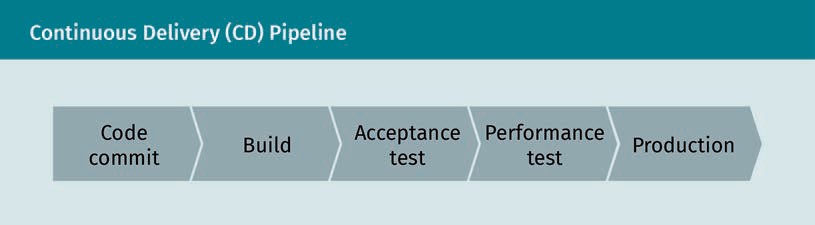
A typical CD pipeline has the following steps (Chen, 2015).

1. Code commit: The code commit step provides quick initial feedback to developers on the code they have already checked in. When a developer checks her or his code into the CI/CD tool, this step triggers automatically and executes the source code and the unit tests. In the case of an error, the pipeline halts and informs the devel- opers about the problem. The developer resolves the issues and checks in the code. After re-running the code and in the case of an error-free execution, the pipeline moves forward to the next step.
2. Build: The build step performs the unit tests again to produce a code coverage report. In this step, the integration tests and various code analyses are executed, and artifacts are generated as output. These artifacts are uploaded into the reposi- tory that manages them for deployment or delivery. All the following pipeline steps will run using these artifacts.
3. Acceptance test: This step essentially ensures that the software satisﬁes all deﬁned user requirements. The pipeline generates the acceptance test environment in this step (a production-like environment in which the software is deployed). This includes provisioning and conﬁguring the servers, deploying the software to the servers, and conﬁguring the software. If the software passes the acceptance tests in this environment, the pipeline proceeds to the next step.
4. Performance test: In this step, the pipeline evaluates how the code modiﬁcations will inﬂuence the software’s performance. The pipeline provides a suite of perform- ance tests, and reports the results.
5. Production: The ﬁnal step is deployment into the production environment.

All the aforementioned steps can be automated by utilizing CI/CD tools like Jenkins. In this case, developers could use Jenkins to describe a pipeline in a Jenkinsﬁle that con- tains different stages. The Jenkinsﬁle also includes environment variables, certiﬁca- tions, and other parameters (Jenkins, n.d.-c).

Code coverage

This speciﬁes how a developed software was successfully exercised during a test by utilizing met- rics like function coverage or line cov- erage.



###### Continuous Training (CT)

Continuous training (CT) is achieved when the ML pipeline is automated. To automate the ML pipeline by feeding new data into the model in production, one needs to imple- ment the following steps into the ML pipeline (Google, 2020b).

* + Automated data validation: This step is performed before training the model to check whether the model should be trained or the pipeline execution should be stopped.
  + Automated model validation: This step is performed using the new data right after the model training. This step is required to validate the model before going into production.
  + Pipeline triggers: The pipeline could be triggered to digest new data by on-demand (manual) execution of the pipeline; on a schedule (daily, weekly, etc.); upon the availability of new data; a degradation in model performance; or a signiﬁcant change in the data distribution.
  + Metadata management: After each execution of the pipeline, some metadata—such as pipeline and component versions, the start and end time and date of the execu- tion, the executor, or the conﬁgured parameters passed to the pipeline—should be stored in the metadata store of the pipeline.

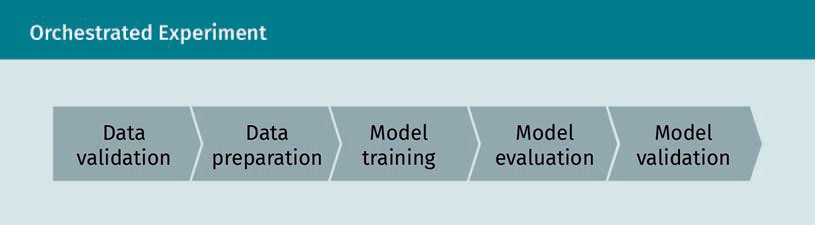
A schematic representation of an automated ML pipeline for continuous training (CT) is shown in Google Cloud, found in Appendix 1 (Google, 2020b).

Here is the list of the ML pipeline characteristics for continuous training (Google, 2020b).

* + Rapid experiment: The transition in pipeline experiment steps (data validation, data preparation, model training, model evaluation, and model validation) should be orchestrated to result in a rapid iteration of experiments.
  + CT of the model in production: The ML model should be trained automatically in production using fresh data.

Softwareentwicklung

* Experimental-operational symmetry: A key aspect of an automated pipeline is to use the same pipeline from the development or experiment environment in the prepro- duction and production environment.
* Modularized code pipeline components: The components of the ML pipelines need to be reusable, composable, and potentially shareable across ML pipelines. There- fore, the source code for components must be modularized.



###### MLOps and ML Pipeline for CI/CD

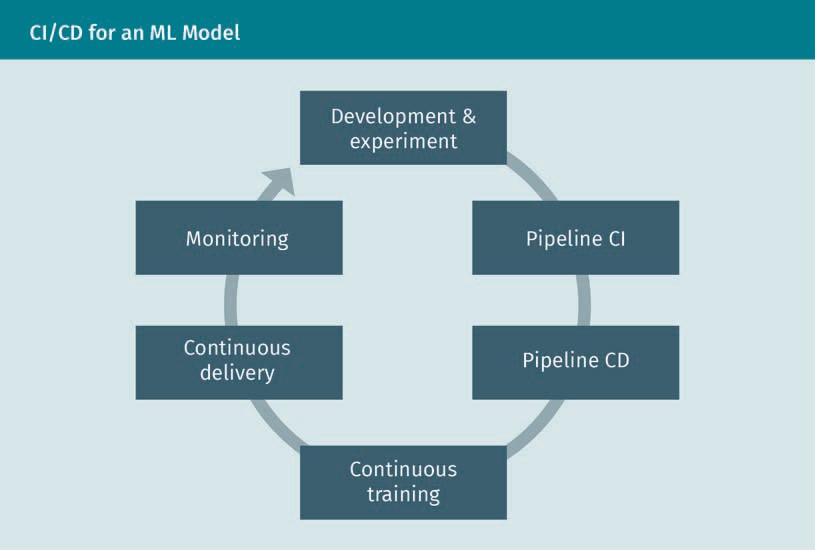
An ML system is also a software system, therefore similar procedures apply for ensuring that ML systems can be built and operated reliably at scale. However, ML systems are distinguished from conventional software systems in a number of ways (Google, 2020b). For example, an ML project team needs data scientists with experience in explorative data analysis, model development, and experiments, but that doesn’t necessarily mean that they have software development skills. The development process is different as well, as ML system development is experimental. This means a developer should try different algorithms, modeling techniques, and parameters to discover the best solu- tion for the problem. In addition to the conventional tests—such as unit and integra- tion tests—data validation and quality evaluation of the trained models should also be tested. The deployment of the ML systems can also differ in that it sometimes needs a multi-step pipeline which adds complexity to the deployment process. Finally, due to the evolving and dynamic nature of ML systems, there are more possibilities for model degradation than conventional software systems.

Due to the above characteristics of the ML systems, there are some differences in CI and CD with respect to the conventional software systems (Google, 2020b):

* In addition to testing and validating code and modules, CI in ML systems is also concerned with testing and validating data, data schemas, and models.
* In addition to a single software or a service, CD in ML systems is about a system (an ML training pipeline) that should automatically deploy other services (like model prediction service).
* CT (cntinuous training) is a new property that is unique to ML systems. CT is mainly concerned with automatically retraining and serving the models.

Considering the above differences, a CI/CD automation of an ML system could have the following six steps, which are also illustrated in the next ﬁgure (Google, 2020b).

1. Development and experiment: By examining different ML algorithms and new ML models, the development team ﬁnds the most appropriate source code for the ML pipeline which is pushed into the source repository as the output of this step.
2. Pipeline continuous integration: The source code from step 1 will be executed and tested. The output of this step is several pipeline components—such as packages, executables, and artifacts—to be used in later steps.
3. Pipeline continuous delivery: The artifacts generated in step 2 (CI) are deployed to the target environment. The output of this step is a deployed pipeline which includes the new ML model.
4. Automated triggering: The pipeline is automatically executed in the production envi- ronment in response to a trigger or based on a schedule. The output of this step is a trained model pushed into the model registry.
5. Model continuous delivery: The trained model is used as a prediction model to serv- ice the customers. The output of this step is the prediction service.
6. Monitoring: The statistics regarding the model performance based on the live data will be collected. The output of this step is a report and also likely a trigger to exe- cute the pipeline for step 1.

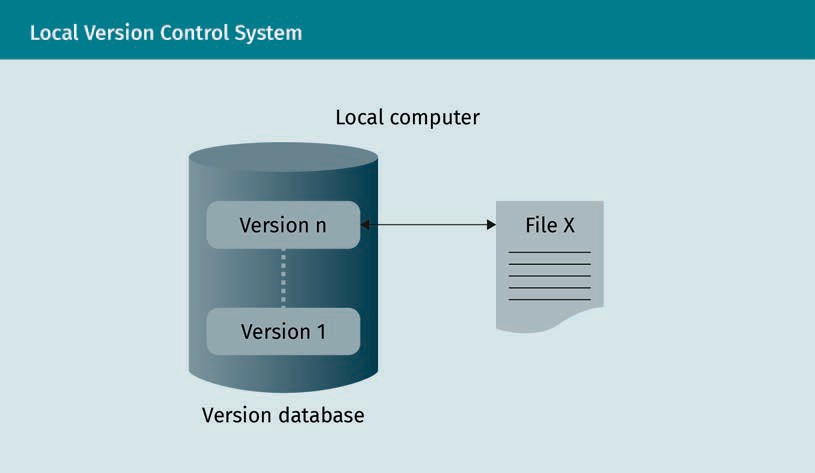


### Version Control

During the software development lifecycle, the development team must keep a histori- cal track of every single change in applications, such as source code, project docu- ments, or build script (especially in the case of a large software project with many developers and multiple teams). The response to this essential requirement has

Softwareentwicklung

already been developed over decades: the version control system (VCS). A VCS also enables different developers and teams within a project to work together in parallel on separate parts of an application while keeping a system of records (Humble & Farley, 2015). Using a VCS, it is possible to revert selected ﬁles to a previous (saved) state or even revert the entire project to a previous state. The ﬁrst version control system was developed by Marc J. Rochkind at Bell Labs in 1972 and was called the SCCS (Source Code Control System) (Rochkind, 1975). SCCS has been followed by many open-source version control systems like RCS (Tichy, 1982), CVS (Price, 2005), Apache Subversion (Free Software Directory, n.d.), and Git (Git, n.d.), as well as many commercial solutions like Perforce (Perforce, n.d.), StarTeam (StarTeam, n.d.), IBM Rational ClearCase (IBM, n.d.-b), Mercurial (Mercurial, n.d.), and Microsoft Teams Foundation System (MS Teams, n.d.). In this section, we will discuss one of the open-source VCS solutions. Because SCCS and RCS solutions are not very common these days, we won’t discuss them here; instead, we will focus on one of the most popular VCSs: Git. Before discussing Git in more detail, we will have a brief look at different possible VCS categories. The ﬁrst is local VCS. This is the simplest type of a VCS system which probably every developer has used at least once in the very early days of practicing coding. It is based on copying modiﬁed ﬁles into a new directory labeled with a date (perhaps a time-stamped directory, if they’re clever). Although it is the simplest approach for a single developer, it is also massively prone to errors like losing track of directories and overwriting ﬁles. To overcome this problem, one can develop a very simple local version database that keeps track of the changes.

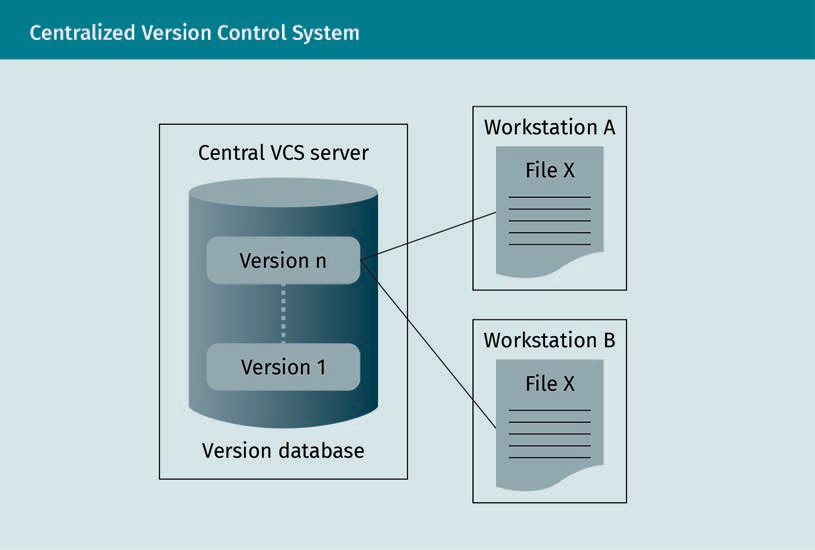


The second is the centralized version control system (CVCS). If more than one developer or development working station is engaged in a project, another VCS solution should be used for this case, as shown the next ﬁgure. In a CVCS, the information regarding the ﬁle changes are stored on a central server and shared with the clients (workstations). Among the available products in this category, one can mention Subversion (Free Soft- ware Directory, n.d.), Perforce (Perforce, n.d.), and CSV (Price, 2005). This system has

Single point of fail-

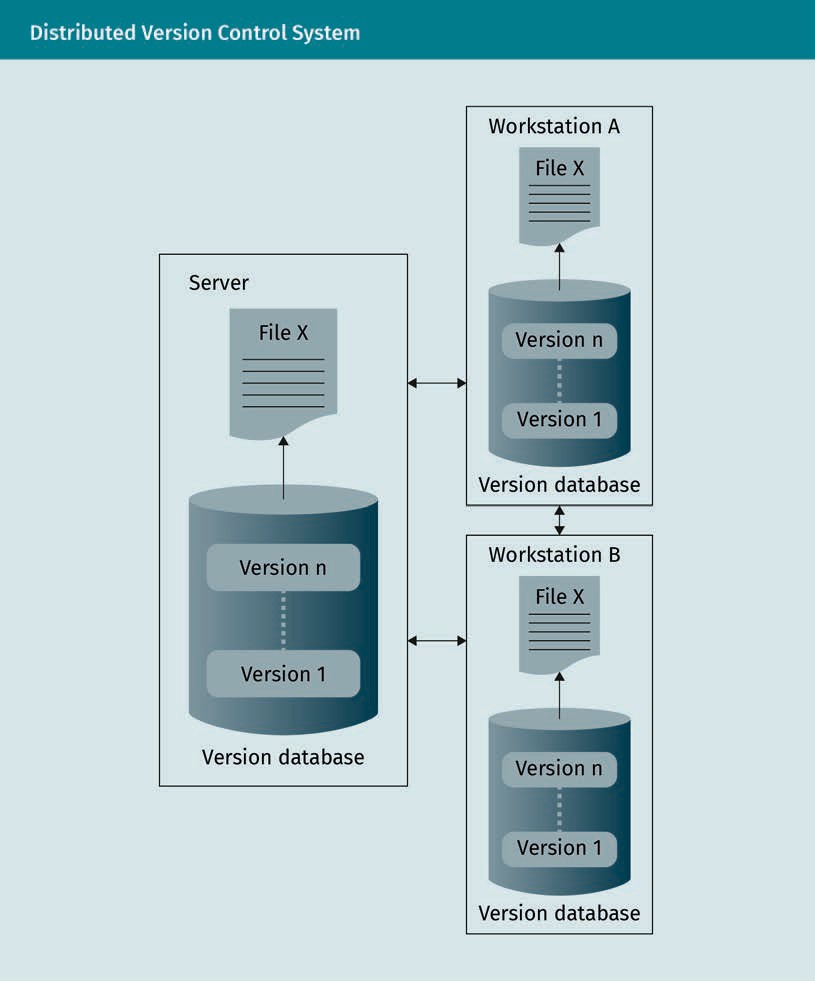
ure Any non-redundant part of a system, the dysfunction of which causes the entire system to fail, is called a SPOF, or sin- gle point of failure (AVI Networks, n.d.).

some beneﬁts, e.g., every team member can see (as far as authorized) what other team members are doing and admins can control the roles. But there are some drawbacks such as a single point of failure, like a central server (if the server fails, workstations lose track of changes).



Lastly, the ultimate solution to resolve the issues associated with the local and central- ized VCSs is using a distributed version control system (DVCS), such as Git (Git, n.d.) or Mercurial (Mercurial, n.d.). In a DVCS, developers do not check out the latest version of the ﬁle, but they also fully mirror the repository. Therefore, the single point of failure problem is resolved because, in a DVCS, any single user’s repository can be copied back to the central server to restore lost data.

Softwareentwicklung



###### Git

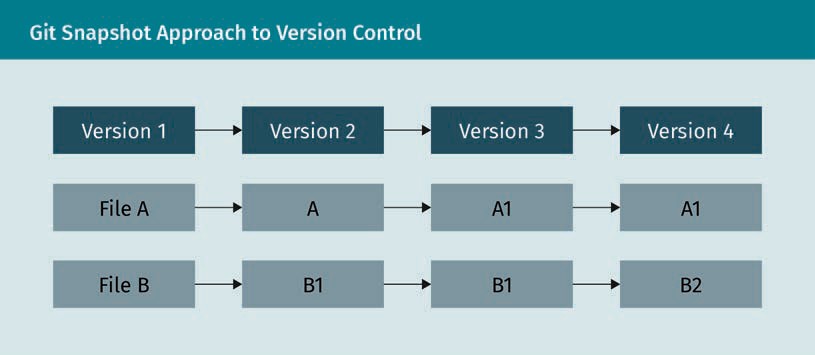
Git project history is entangled with the Linux Kernel project, an open-source software project. In 2005 the Linux Kernel team decided to stop BitKeeper (a distributed VCS known as DVCS) as the version control system and develop its own VCS, a decision that led to the birth of the Git project (BitKeeper, n.d.). Git is a VCS that is simple but fast with strong support for non-linear development. It features thousands of parallel bran- ches, fully distributed, and capable of managing large projects. Unlike most VCSs that store modiﬁcation information as a list of ﬁle-based changes (i.e., storing a set of ﬁles and the related changes), Git stores a snapshot of every state of the project right after

Branch A branch is a copy of the main repository of a version control

system.

Checkout Checking out a branch updates the ﬁles in the working directory to match the version started in that branch, and it tells Git to record all new commits on that branch (Atlassian Bitbucket, n.d.).

a user commits (saving changes into the local repository). To be more efﬁcient, if a ﬁle has not been changed, there is just a link to the previous state of the ﬁle in the new snapshot. In this regard, we could consider Git as a mini ﬁle system (Git, n.d.).



One advantage of Git is checksum. For every change, a checksum is computed before it is stored, and it is impossible to modify anything without Git knowing about the modiﬁ- cation. This feature prevents users from losing information in transit. To implement the checksum feature, Git uses SHA-1 hash, a 40-character string composed of the hexade- cimal characters (Git, n.d.). This hash is calculated based on the ﬁle content or directory structure in Git.

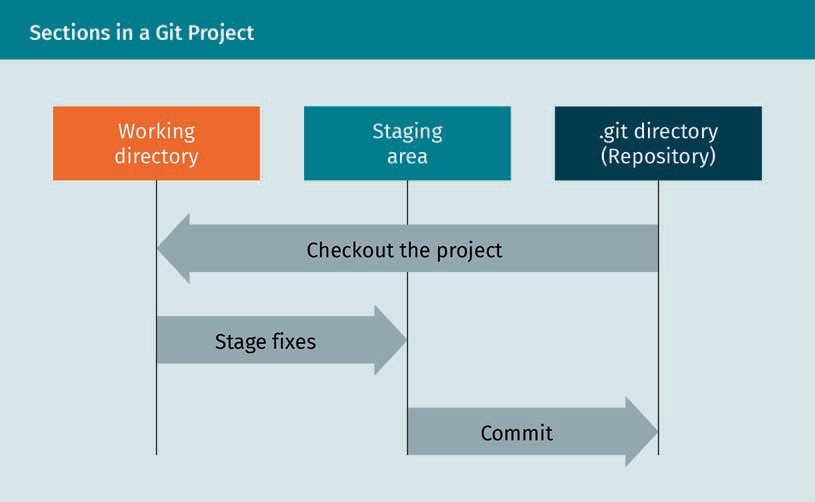
There are three different states that a ﬁle in Git can have.

1. Modiﬁed: The ﬁle has been changed but the user has not committed the changes to the database. A ﬁle in this state is marked as modified.
2. Staged: A modiﬁed ﬁle has been marked by the user to go to the next commit snapshot. A ﬁle in this state is marked as staged.
3. Committed: The data have been saved on the local database safely. A ﬁle in this state is marked as committed.

Based on these three states, a Git project is composed of three main sections or stages. The ﬁrst is the working directory, which is a single checkout of a version of the project. The ﬁles in this directory are pulled (incorporates changes from a remote repository into the current branch) from the Git repository. The second is the staging area. This is a ﬁle in the Git directory which contains information about the changes which are going to be committed in the next commit. Thirdly and ﬁnally, we have the Git directory. This is the most important part of Git that stores the metadata and object database of the project. When the user clones the repository from another workstation, it is the Git directory that is being copied. This process can be seen in the next ﬁgure. Regarding these sections of a Git project, a simple Git workﬂow has the following steps (Git, n.d.).

Softwareentwicklung

* 1. The user modiﬁes the ﬁle in their working directory tracked by Git.
  2. Then the user stages the part of the changes that should be committed in the next commit in the staging area.
  3. The ﬁnal step is to commit. This takes a snapshot of the ﬁles from the staging area and stores the snapshot in the Git directory.



Below, you can see a practical example of a Git Workﬂow (Dudler, n.d.).

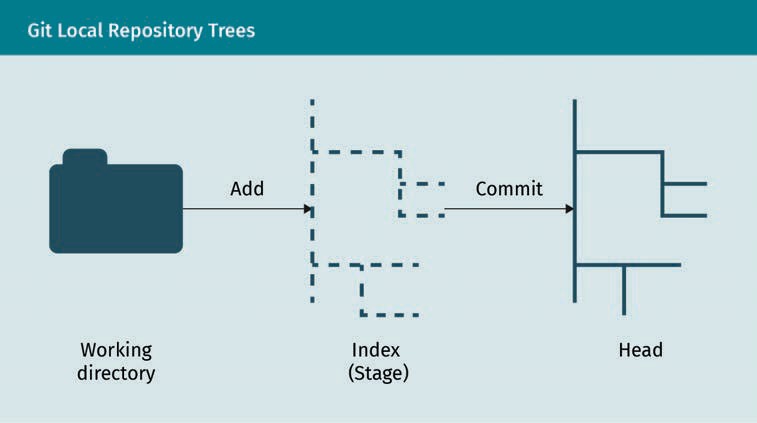
1. Create a new repository in the project directory.

git init

1. Create a working copy of a local repository by running the command.

git clone /path/to/repository

1. The local repository consists of three “trees”:
   1. Working Directory contains the actual ﬁles.
   2. Index is the staging area.
   3. HEAD points to the last committed version.



1. Propose changes (add it to the Index).

git add <filename>

1. Commit these changes into the Head.

> git commit -m "Commit message"

1. Send changes to the remote repository.

git push origin master

1. Create a new branch called “branch\_1”.

git checkout -b branch \_1

1. Switch back to the master (main) branch.

Git checkout master

1. Push the branch to the remote repository.

Git push origin <branch>

1. Update the local repository to the newest commit.

Git pull

###### GitHub

GitHub is a cloud-based developer platform based on Git (GitHub, n.d.-a). The funda- mental difference between Git and GitHub is that Git is an open-source tool where the developers can manage their source code locally, while GitHub is a cloud-based plat- form where developers can share their projects with a community, have access to developer tools and review each other’s code, among other things. Like Git, at the core of a GitHub project is a project repository. A project repository contains all components that a project needs: ﬁles, folders, spreadsheets, data sets, etc. It normally also con- tains a README ﬁle, which contains information about the project. By default, the repo-

Softwareentwicklung

sitory has one branch, called master. Branching is an approach to work on different versions of the repository at one time. The branches are used to test the changes in the source ﬁles before committing them to master. When a developer is working on a branch of the master, if there are some changes in the master, the developer of the branch could pull in the changes from the master.



### Entwicklertools

Programming tools or software development tools are computer programs that support software developers and the development team to realize software projects. These tools are utilized to create and modify the source code of programs with text editors; get help in their program ﬂow from special graphical user interface editors; translate their source code into an executable machine language with compilers or assemblers; test and debug developed solutions with test tools or debuggers; and store and man- age programs and program documents with version control systems. In this section, we will review some of the most common development tools.

###### Kommandozeile

The ﬁrst, most straightforward, and most common development tool is the command- line interface (CLI). The command-line interface is a part of a computer program that receives a line of text as input from the user and interprets it (using the command line interpreter) in the context of a given operating system or programming language (Kumar, 2016). Operating systems and programming languages implement the com- mand-line interface into a shell to access the operating system or write code. Examples of such CLI are Unix shell (Bourne, 1978), PowerShell (Bright, 2016), Z Shell (Z Shell, n.d.), and Python Shell (Python Shell, 2020).

A command in an operating system CLI could have components such as

prompt command parameter\_1, …, parameter\_n

Shell

In computer science, a shell is a software by which a user interacts with an operating system or program.

Here, the prompt provides the context for the user (and usually ends with one of the $,

%, #, :, > or – characters). The command is provided by the user to execute a special task, and parameters are the optional parameters provided by the client to control or limit the command. Although the CLI is the most straightforward way to communicate with an operating system or writing code, it is not the most suitable solution in some cases, especially for beginners, for in-line editing and for debugging. However, there are some integrated tools in each CLI which can make life easier for developers. There are many examples such as Vim text editor for Unix and Apple OS CLIs (Vim, n.d.); Wget, for retrie- ving ﬁles using HTTP, HTTPS, FTP, and FTPS via the web (Free Software Foundation, n.d.- a); and Gzip, a data compression program (Free Software Foundation, n.d.-b).

###### Integrierte Entwicklungsumgebung (IDE)

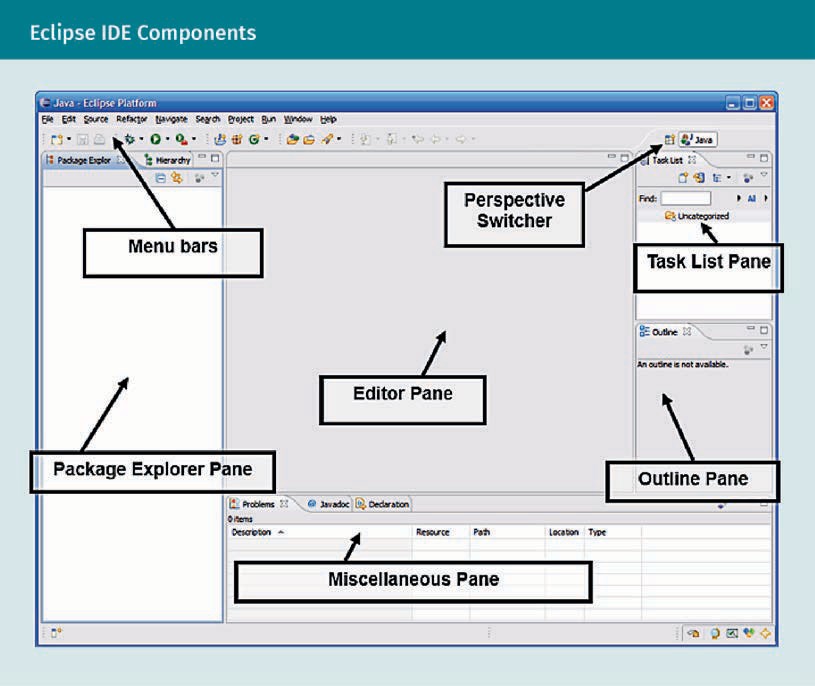
An integrated development environment (IDE) is a single program or platform that comprises several developments that facilitate software development procedure. Exam- ples of IDEs are Microsoft Visual Studio, Eclipse, NetBeans, and PyCharm. Typical func- tionalities of a modern IDE include

* intelligent code completion. This is a convenient method to access functions’ descriptions and their parameter lists to speed up software development. In this method, after typing the name of a function in the IDE, a short description of the function together with a list of the input parameters of the function will appear. An example of such functionality is IntelliSense, which is a combination of code editing features including code completion, parameter info, quick info, and member lists (Visual Studio, n.d.).
* source code editor. The developer can use the IDE to write and edit the source code. Some features such as intelligent code completion, will facilitate the edition proc- ess.
* build automation. IDE can automate the build process, including compiling the source code, packaging the compiled ﬁled into a compressed format, and producing installers.
* debugger. An IDE debugger lets you change the value of the variables at run-time, see the value of the variables also at run-time, break execution at any point in the code, etc.
* Syntax highlight. An IDE for the supported programming languages highlights the test with different colors and fonts as shown the following ﬁgure.

Softwareentwicklung



In the following, we will review one of the most common open-source IDEs: Eclipse (Eclipse, n.d.). Eclipse started as proprietary technology, led by IBM, and in 2001 the Eclipse open-source project was announced by the initial Eclipse Consortium (Eclipse, n.d.). It provides tools for coding, building, running, and debugging applications and was originally designed for Java. Eclipse now supports many other languages such as C, C++, Python, PHP, and Ruby (University of Maryland, 2018). The components of the Eclipse IDE are shown below.



* Menu bars: full drop-down menus plus quick access to common functions
* Perspective switcher: to switch between various perspectives
* Package explorer pane: where our projects/ﬁles are listed
* Editor pane: the place to edit the source code
* Miscellaneous pane: several components are listed here such as a console and a list of compiler problems
* Task list pane: a list of “tasks” to complete
* Outline pane: a hierarchical view of a source ﬁle

###### Jupyter Notebook

Jupyter Notebook is a web-based interactive programming application that is used for developing, documenting, and executing code, as well as communicating the results (Jupyter Team, n.d.). Jupyter Notebook is mostly used by data scientists to run Python codes, but it supports about 40 different programming languages. In general, it can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter Notebook combines two functionalities. Firstly, it includes a web application (or a browser-based tool) for inter- active management of documents that combines descriptive text, mathematics, com- putations, and their media output, such as diagrams or 3D-visualisations. The web

Softwareentwicklung

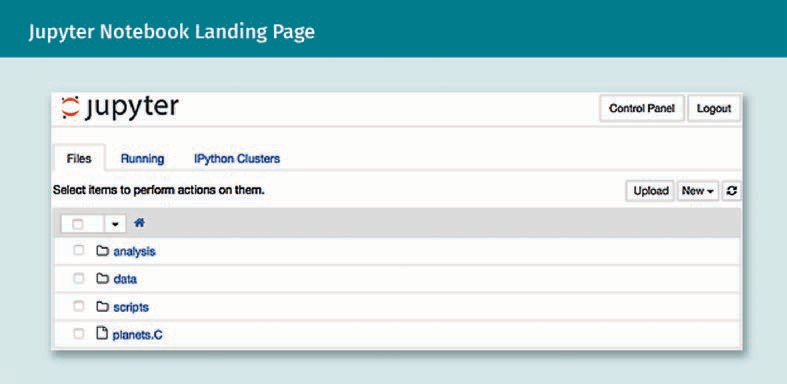
application makes it possible to edit the text in a browser with automatic syntax high- lighting; execute the code from the browser; represent the result of computations in PNG, SVG, HTML, and other rich media representations; and include mathematical nota- tions using LaTeX. Secondly, it includes notebook documents which are a representa- tion of all content visible in the web application. These include inputs and outputs of the calculations, descriptive text, mathematics, images, and media representations of objects, as well as the source code. Such documents are stored in a JSON ﬁle with the .ipynb extension. Storing as a JSON ﬁle facilitates the version control of the code. In addition, any .ipynb ﬁle which is available on a public URL can be shared easily (Jupyter Team, n.d.).

Using Jupyter Notebook

1. One can start a notebook server from the command line using the following com- mand (Jupyter Team, n.d.):

>jupyter notebook

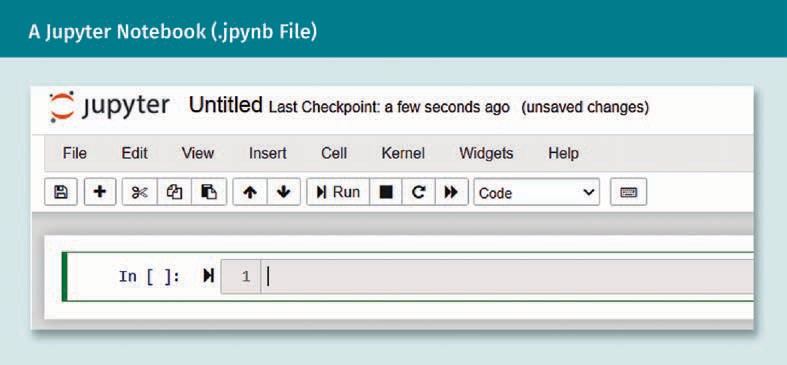
1. The landing page of the Jupyter Notebook web application (by default, http:// 127.0.0.1:8888), which is called the dashboard, includes the available notebooks in the notebook directory.

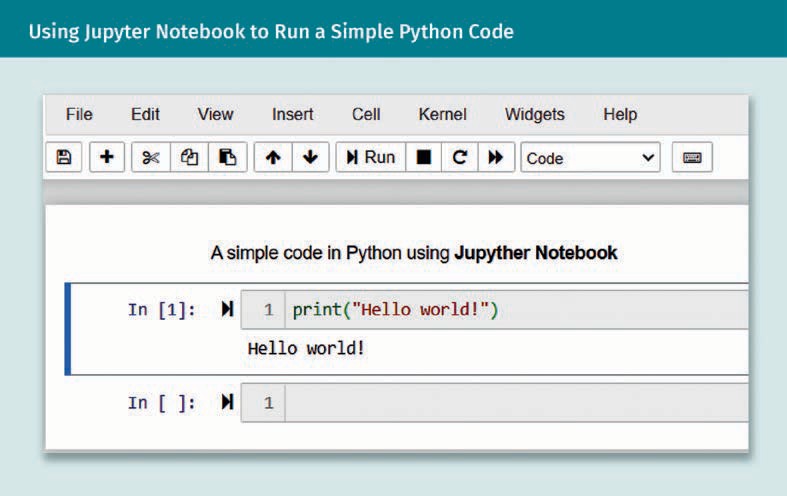


1. The user can then create new notebooks from the dashboard under new/notebook. The new notebook (a .ipynb ﬁle) is shown in the next image.

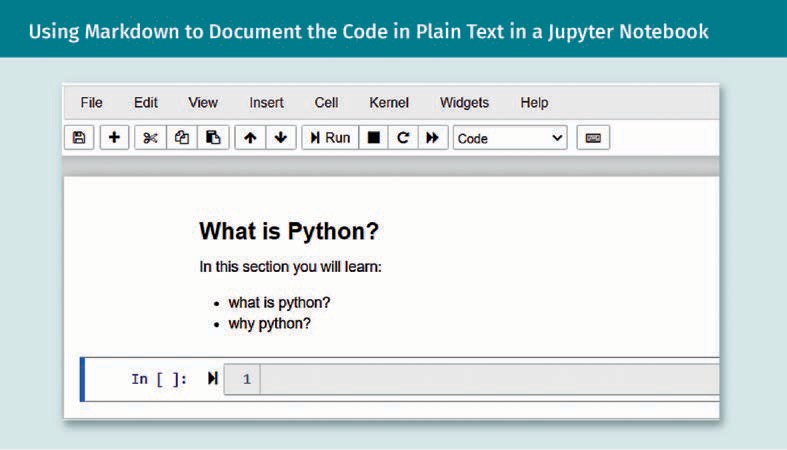
JSON ﬁles

These are pure text ﬁles that are only intended to exchange data bet- ween different appli- cations.

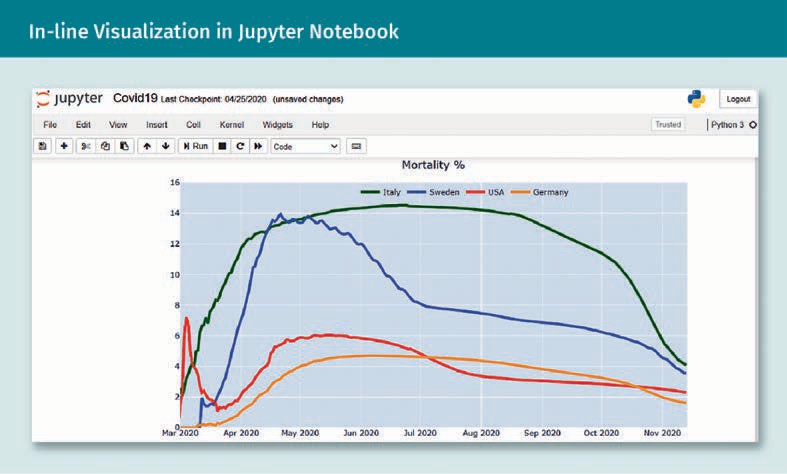


1. After this, the user can start to write code inside the notebook.
2. Users can document the code in plain text using the markdown language. It also supports text markup (italics, bold, form lists, etc.) as shown in the following ﬁgure.

Softwareentwicklung



1. One can also use Jupyter Notebook to plot in-line, as shown below.



Zusammenfassung

In this unit, we discussed different approaches to software development with an emphasis on testing. We began by discussing different test scenarios such as unit, integration, system, functional, and non-functional tests (categorized by Marick’s quadrant). Then we discussed how to test a machine learning system as a non- deterministic model, as well as how both data and learning programs from ML models should be tested. Three approaches to test in parallel to developing were introduced: test-driven development (TDD), behavior-driven development (BDD),

and acceptance test-driven development (ATDD). After this, we learned how to automate the pipeline of project development using continuous integration (CI), continuous delivery (CD), continuous testing (CTe), and continuous training (CT) in the case of machine learning pipelines. For the CI/CD in a machine learning pipe- line, we learned that there are six steps: development and experiment, pipeline CI, pipeline CD, continuous training, continuous delivery, and monitoring. We were then introduced to the principles of version control systems for the software projects and explained two common solutions: Git and GitHub. Finally, we learned about the development tools such as command-line interfaces (CLI) and integrated development environments (IDE).



# Lerneinheit 4

## API

#### LERNZIELE

Nach Abschluss dieser Lerneinheit sind Sie in der Lage...

… the various ways of interacting with software and services.

… the most important principles to know when designing and building interfaces.

… how to distinguish good from bad interface design.

… how to build a Python library with good design in mind.

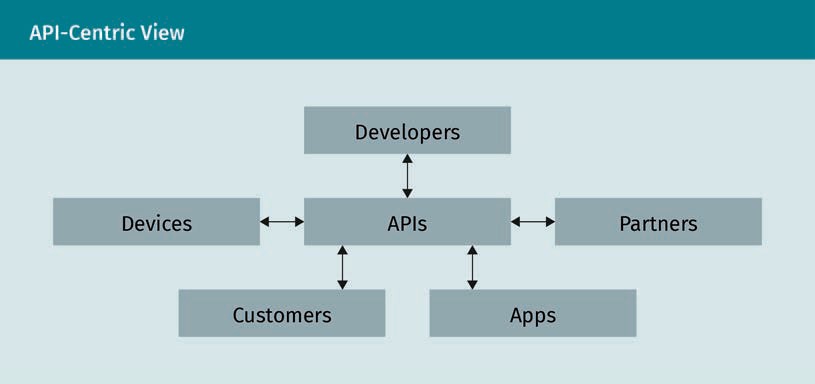
DL-E-DLMDSSEDIS01-U04

1. API

### Einführung

In an increasingly digitalized and interconnected world, a large part of the world popu- lation has access to smart devices and uses them for an ever-growing range of use cases. That means that even technically inexperienced or inapt users cannot help but use complex software on a regular basis. To make this work, the ways in which you interact with your computer must be well-designed and easy to grasp. Not only that, the interface you’re using to interact with any piece of software must be stable and reliable, so that the services you’re enjoying don’t surprise you or suddenly stop work- ing. Take Amazon’s retail business as an example. Chances are you’ve seen an e-com- merce platform like Amazon, browsed for products, selected a few of them and put them in your cart. The website you intuitively interact with is just the user interface to a massively complex software and hardware backend powered by tens of thousands of machines scattered around the globe and operated and maintained by an army of engineers who built that platform in the last two decades. Since it’s so common today to use services and check apps, it’s also easy to forget about the typical discrepancy between the way you interact with software and the rest of it. Your interaction with apps can be seen as the ﬁgurative tip of the iceberg of software complexity, and while some websites might be overwhelming, or simply poorly designed, you can rest assured that the interface you’re using is as much about what you need to know as it is about all the things you mustn’t be exposed to. However, the story doesn’t end at the level of clicking through user interfaces. All software engineers and data scientists interact with computers every day, and while their interaction looks distinctively different from that of your average browser user, they nevertheless still have to rely on contracts to intero- perate with their programs and tools: These contracts are called application program- ming interfaces (APIs). The term API is increasingly known by non-professional tech enthusiasts, and, to stick with the example of Amazon: Exposing their internally devel- oped tech stack through APIs in Amazon Web Services (AWS) has become a business line of the same order of magnitude as their retail endeavor. While not every click on a website is an API call, every business-relevant action a user carries out corresponds to one. In fact, one can take an API-centric view of modern, digitized business interac- tions, in which not only API developers, but also apps and devices, as well as your cus- tomers and strategic partners, all interact through APIs.

API



This unit is devoted to giving you an overview of common API paradigms, that is, the multitude of ways to interact with computers in a productive way. The focus for this unit is very much on a speciﬁc group of users, namely engineers and scientists, power users like you. Apart from being aware of what you’re using and how you’re using it, as an experienced professional it is crucial to develop a mindset of evaluating interfaces critically. In the end, a lot of the code you produce on the job will be an API for some- one else to consume, so it’s worthwhile to develop the skillset for building great inter- faces yourself. Being on the producing or consuming end of an API is just two sides of the same medal. Therefore, we’re going to build out a few examples of APIs in Python to demonstrate what good interface design looks like in practice, while also showing you counterexamples to learn from. As Martin Fowler puts it: “[a]ny fool can write code that a computer can understand. Good programmers write code that humans can under- stand” (Fowler & Beck, 1999, p. 15).

### Interacting with Software and Services

Modern computers have an incredibly high level of complexity. Even if you consider yourself an expert, the only reason you’re capable of being productive in your interac- tion with your machine is that you rely on the existence of many layers of abstractions that smartly hide what’s going on under the hood. For instance, if you’re doing a data analysis in Python, you want to focus on your speciﬁc use case and not how your code translates into zeros and ones, or how exactly your CPU is utilized by your operating system to run this analysis. Ultimately you have to trust that your code, a tiny fraction of all programming capabilities, does what you intend it to do. In other words, you’re constantly using application programming interfaces (APIs) to shed complexity and be productive. As alluded to in the introduction, an API is an interface that allows the interaction between one or several parties and a software component. To be more pre- cise, it is key that an API deﬁnes the requests or calls you can make with it, clearly states the data formats it expects as input and what formats it produces, and communicates which conventions it follows and what the intended behavior of each request is.

To give you a ﬁrst, concrete example of a class of API calls, consider Python’s function- ality to import libraries. It is centered around a single keyword: import. There are three essential calls you can do with import, namely, importing Python modules, importing functionality from a module, and aliasing. Throughout this unit we’re using Python 3.7 or later for our examples. Here’s an example:

import time # import the time module

from time import ctime # import the “current time” import time as t # refer to time module as t

To use import, you always have to follow it up with a valid Python package name, i.e., either a built-in package or a third-party package installed in your Python environment. An import statement does not return anything if successful (we say it is silent), but fails with an error message if the imported library does not exist. For instance, if you type import testfailure into a Python session, you will see the following error mes- sage:

ModuleNotFoundError: No module named ' testfailure '

In terms of intended behavior, unsurprisingly, the role of import is to import and make Python functionality accessible to users in their current session or script execution. For example, the above import statements would allow a user to query the current system time with ctime(), which is returned to the user as a Python string. Before we dive into more aspects of APIs and examples of types relevant for your work and studies, let’s have a look at the evolution of the term API and what it means today.

###### Historical Evolution of APIs

The term API has undergone an interesting evolution since its inception. From today’s perspective, programmers have arguably been building APIs ever since the ﬁrst pro- grams have been written. Originally, “API” was only used for user-facing applications, but quickly expanded from there. The term itself was formally published in the 60s (Cotton & Greatorex, 1968), but it took some time before the concept was popularized. Cotton and Greatorex (1968) proposed a consistent application interface written in For- tran, intended to make the graphical application they were concerned with easier to use by making it run on any kind of hardware. Hardware independence is crucial to dis- tribute your software across many different machines, something that is obvious to modern application designers but was difﬁcult to achieve in the early days of comput- ing. In 1974, the term API was expanded to database management systems, but the application interface was strictly separated from other ways of interacting with your database, such as querying (Date, 2019). This turned out to be too strict, and later gen- erations realized that by expanding your deﬁnition of an application interface, you

API

could unify all sorts of interactions. Building rich, integrated interfaces thus became the norm throughout all ﬁelds of computing and not only in database design (Berg et al., 1981). By the 90s, it was universally accepted that the term API would be more broadly used to cover all kinds of programming and deﬁned as “a set of services availa- ble to a programmer for performing certain tasks” (Malamud, 1990, p. 294). With the inception of the internet, web APIs became a new extension of the term “network- based Application Programming Interfaces” (Fielding, 2000). Web APIs are so common today that people use “API,” historically a somewhat restrictive term, to refer to web API and their communication protocols. Most of the time it is understood in the even nar- rower sense of JSON or XML-based web APIs. Throughout this chapter we take a more holistic view on application programming interfaces, of which web APIs are just a part. The quick adoption of the internet and increased interconnectivity of users throughout the world led to an explosion of web APIs which shows no sign of slowing down. But their beginnings were modest. Early pioneers of commercial use of web APIs include dominant forces from the dot-com era, including Salesforce, Ebay, and Amazon. For instance, in 2002 Amazon launched its Web Services (AWS), which allowed developers to embed content from Amazon into their own websites. AWS has vastly evolved since then and Amazon’s cloud offering is almost on an equal footing with its retail business. The next wave of web APIs, starting around 2004, brought us social media integrations with tech giants such as Facebook, Twitter, or Flickr. Twitter published its developer API in 2006 to give engineers access to all kinds of data from the platform. In the same year, Facebook launched its API to let developers access friend information, photos, posts, etc., which had a high impact on attracting advertising companies and helped Facebook become a global player in the social media industry (Lane, 2019). In yet another wave, companies began to move their infrastructure and services to the cloud, pioneered by Amazon in 2006 with a new batch of services, led by Amazon Simple Stor- age Service (S3) and Amazon Elastic Cloud Compute (EC2), both of which are still ele- mental to AWS’s success story. While S3 made cloud storage accessible with a simple pay-as-you-go approach, EC2 gave developers access to compute resources in the cloud, spinning up their own machines and hosting their services there. It’s interesting to acknowledge the fact that the success of AWS is partially credited to Amazon’s API- oriented culture (API Evangelist, 2012). Internally, to this day, all Amazon teams have to expose their functionality as services for other teams to use. While this philosophy might have incurred some overhead in the short-term, as building APIs is expensive, the genius move of exposing the company’s infrastructure as a service to the public would not have been possible without it. With the release of the iPhone, the internet has become more mobile and so did the next generation of APIs. To brieﬂy mention a last notable shift in the tech landscape that had an impact on APIs, with the emer- gence of the Internet of Things (IoT), more and more devices are now connected to the internet and eligible for API consumption. Examples include Amazon’s Alexa (2014), which allows API access for each notable feature, such as voice interaction, alarm set- ting, or providing weather information. Fitbit (2017) produces ﬁtness-metric-tracking wearables powered by APIs. In a historical context, nothing speaks more for the impor- tance of APIs and the commercial interests they represent in this modern era than the infamous case of tech giants Oracle and Google ﬁghting over copyright issues of 37 Java APIs in 2017 (Tsidulko, 2020). To conclude, today’s APIs and their usage are ubiquitous and found in many areas relevant to you:

* + - APIs are behind practically all web applications.
    - They power mobile and desktop applications.
    - APIs underlie all network communication between the devices and applications on the internet.
    - They are a cornerstone of the Internet of Things (IoT), connecting smart devices such as automobiles, fridges, or vacuum cleaners.

###### Speciﬁcations and Contracts

Every piece of software has a provider, the company or group of people who authored it, built it, and currently maintain it, and consumers, who utilize the software to cater to their needs. Consumers can either be humans or other machines using the software. Regardless of the actual roles, an API can be seen as a speciﬁcation or contract bet- ween providers and consumers. As with any other contract, the parties involved must ﬁrst come to an agreement regarding what the contract is about and then write it down. In software, the provider can dictate what an interface looks like to a certain degree but must make sure not to lose its customer base, for instance by sticking to bad design. By the time an API, which is viewed as a contract, is agreed upon, it is binding for all par- ties. Companies providing software-as-a-service (SaaS) usually have service level agree- ments (SLAs) that include API stability. In particular, a company can’t change the struc- ture of its API calls over night. This perspective implies certain problems when it comes to the maintenance and extendibility of APIs. Not unlike software packages or operating systems, APIs come with versions attached to them. For instance, the developer plat- form GitHub has a developer API, the URL of which reads

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

Note the v3 sufﬁx in the URL: This is the current stable version of GitHub’s API, but his- torically it is the third major version of their API. The ﬁrst two have been deprecated and removed years ago. The beneﬁts of API versioning are manifold:

* + - Within one version you guarantee that every single API call has a stable interface. The consumers of the API building downstream applications can rely on it to not stop working without fair warning.
    - A new functionality can be added to an API version if it is independent of old requ- ests. In software releases this would correspond to minor releases or patches. Alt- hough bug ﬁxes are part of keeping an API alive, this type of API extensibility can be very convenient within a release.
    - New API versions are reserved for fundamental changes. We speak of breaking chan- ges if code that consumes an API in an older version stops working with the new one. API developers should try to get interfaces right ﬁrst-shot, but of course that’s not always possible. It is still important to think about user-facing API design decisi- ons carefully, as they are more difﬁcult to ﬁx than internal processes. Versioning APIs allows providers to roll out several versions of their API at the same time and gives consumers a fair warning of how long the old version will still be alive and how to migrate gracefully to the latest. Giving users time to react to changes and adapt is a massive beneﬁt as opposed to making abrupt changes over night.

API

* Communicating deprecation warnings for older API versions while still maintaining them is an added burden on providers, but it is much more convenient for consu- mers.
* Old API versions can be deprecated when they become buggy, highly inefﬁcient, or encourage bad coding practices. Individual calls can be removed entirely in a new version if they’re not needed anymore.

Dokumentation

A topic that deserves special attention in the context of API contracts is documenta- tion. While often seen as an unwanted but necessary evil on the part of application developers who want to focus on their code, documentation is crucial to enforce a common understanding of an API. In fact, no matter how simple you think an interface is, in the end a human will have to operate it (if only to instruct a machine to use it) and there’s always a chance for miscommunication. Some go as far as to say that the documentation is the API and the code just its implementation. From a software plan- ning perspective this is not an unusual thought. Before the advent of the Agile Mani- festo, it was customary in all engineering ﬁelds to write explicit, extremely detailed spe- ciﬁcations (or specs) that had to be adhered to before any of the coding or building could begin. A spec—for an API or another product—can be seen as the result of a potentially long negotiation process that in project management is often described as requirements engineering. From a pragmatic point of view documentation is not to be underestimated. In an API variant of Murphy’s Law, you could say that any API call that can be misunderstood will be misused, so it’s worth being as clear and precise as you can when building and documenting interfaces.

###### Security, Governance, and Release Policies

APIs allow companies to integrate with technology outside of their corporation, which introduces dependency. Likewise, when a company offers access to an API, it suddenly must deal with external dependents. Both sides of this coin do not only imply gover- nance issues, but also potential security risks. On the governance side, if your business relies on the availability of an API and that API slows down, it will slow down your busi- ness, too. So, the performance monitoring of that external API becomes an integral part of your internal business processes. On top of that, if an integration suddenly stops working, you must ﬁgure out quickly whether it’s due to a change on your side or from the external provider. That means that your internal development cycle must consider the external API development cycle that you rely on. It is not uncommon for larger companies to have a dedicated API manager who plans and manages all (external) API- related topics within a company, such as making sure the integrations comply with organizational or governmental standards. On the ﬂip side, if you’re exposing an API to the public, chances are that this API accesses production data of your company, which can potentially be compromised. For instance, as a health care data provider in most countries, you must keep your data anonymized at all costs. The security risks for com- panies with sensitive data can be very high, so investing in sufﬁcient security standards and setting proper access rights for your APIs is crucial. On a high level, there are three main policies which APIs can be released under (Boyd, 2014).

Deprecation

An API can declare some of its calls as deprecated, which means that this part of the API is either modiﬁed or remo- ved. Deprecation warnings give deve- lopers the chance to transition to parts of the API with long- term support.

Published vs. public

interfaces Only if you publish an API, as in you expose it publicly on the internet, does it truly become public in the sense dis- cussed in this unit, with all beneﬁts and

drawbacks.

POSIX

The portable opera- ting system inter- face, or POSIX, is a set of standards speciﬁed by the IEEE Computer Society.

POSIX deﬁnes an API on the operating system level, inclu- ding pipelines, proc- esses, shells, and utilities, among many others.

* Private: Your API is only used internally.
* Partner: You expose your API only to a speciﬁc group of users, namely, speciﬁc busi- ness partners.
* Public: Your API is publicly available. In typed programming languages you often encounter the keyword “public” for a class or function, which means that this part of the respective library is available to all programmers working with that library. How- ever, this kind of “public” functionality can still be part of a private API inside a com- pany. Note that “public” is not the same as free or available to everyone, it just means that if you’re granted access and pay for it, you can use it (see Fowler, 2002).

One of the advantages of versioned APIs we discussed was interface stability within one release of an API. A common practice among API designers to grant them some ﬂexibility is to mark their calls as unstable. Google’s Java library Guava marks unstable components with a @Beta annotation, indicating that you can’t rely on it, as it might change soon or vanish altogether.

###### API Types

Now that you’ve seen some examples, learned about the history of APIs, and under- stood some of the key aspects of what makes and breaks an API, let’s have a look at various levels and types of APIs. You will notice that we’re working “inside-out” from the lower-level components of a computer to higher-level application design. This onion-like layering of API abstractions is what makes it possible for you to work efﬁ- ciently with a computer. Writing a script like the one you’re reading right now was done on a text processing system without thinking for a second about the hardware that’s running it.

Operating systems

Simply put, an operating system (OS) is software that manages and orchestrates hard- ware components and software resources. It also provides common system-level ser- vices for computer programs run by its users. The POSIX standard originated from work by Richard Stallman in the 1980s to unify different UNIX operating systems under one umbrella and is the foundation of the Linux operating system (Stallman, 2011). POSIX speciﬁes a set of APIs that POSIX-compliant operating systems have to implement, so that compiled programs based on these APIs can run on any other system running POSIX. Some POSIX aspects you probably interact with daily when using a Linux system are its standard for ﬁles and directories, pipes, and speciﬁcations for input and output (I/O Port Interface and Control). Operating systems are very complex pieces of software, and there is a lot more to say about them, in particular about the topic of APIs, but that’s beyond the scope of this script. Other examples of notable APIs that work close to the operating system level and are relevant for data-intensive sciences include

API

* + OpenGL (Open Graphics Library), a cross-language and cross-platform API for hard- ware-accelerated rendering of 2D and 3D vector graphics, which is extensively used in video game development, computer-aided design (CAD), and virtual reality appli- cations. First released in 1992, the API standard is now maintained by non-proﬁt consortium Khronos Group.
  + OpenCL (Open Computing Language), a framework and API for writing applications independently from speciﬁc platforms and for leveraging various hardware compo- nents such as central processing units (CPUs), graphical processing units (GPUs), and ﬁeld-programmable gate arrays (FPGAs). OpenCL provides C/C++-based pro- gramming languages to users and comes with API calls for parallel processing on devices. Like OpenGL, OpenCL is maintained by the Khronos Group consortium. There are OpenCL-compliant implementations for hardware of all major hardware providers, except for Apple, who deprecated OpenCL support in favor of another standard.
  + CUDA (compute uniﬁed device architecture), an API for parallel computing on GPUs developed by Nvidia for its own hardware. CUDA works for instance with C/C++ or Fortran, and while not trivial to program, it is considerably more convenient when compared to OpenGL. Efﬁcient and fast implementations of matrix operations, such as Matrix multiplications and higher-dimensional equivalents thereof have been one of the core drivers behind recent successes in the ﬁeld of deep learning.

Command line interfaces

If you’re regularly using UNIX systems such as Linux or MacOS, but also if you’re a Windows power user, you will likely execute a fair share of your programs in a com- mand line shell or just shell. Tools run on the shell are referred to as command-line interfaces (CLIs) and are a special type of API leveraging OS resources. CLIs are simple yet powerful text-based systems that read instructions line-by-line. Among the many CLI tools available on various operating systems, two tools deserve special attention for Python programmers, namely, python itself and the package manager pip. While you can start an interactive Python session by just typing python into a shell on a system that has Python installed, you can also use it to start Python programs or execute strings interpreted as Python code directly from the shell. Here’s an example of printing the number 42 as output to your shell using the python CLI:

python -c "print(42)"

The -c ﬂag allows you to pass any string that can be read by the Python interpreter. Another example of using python as a CLI API is to query the version installed on the system with python –V. To install packages for Python the standard way is to leverage the Python Package Index (PyPI) via the CLI tool pip (although there are other tools, like the old easy\_install, the fashionable poetry, or the batteries-included conda).

To install the popular pandas library for data analysis you can use

pip install pandas

Updating pip itself, which is just a Python package on PyPI, works by invoking

pip install --upgrade pip

Shell

A shell is a com- mand-line interpre- ter which has access to the API of your operating system.

While it refers to gra- phical interfaces as well, the term shell is mostly used to refer to command- line interfaces (CLIs). The naming conven- tion comes from the fact that shells represent the outer layer of an OS.

As an aside, note that in the context of web APIs you often hear developers talk about having an API and a CLI for a given service. In that case, “API” usually refers to some sort of web interface or access through a programming language, while the “CLI” is the part of the API that is accessed through a shell.

Programming languages

We have just seen how Python can be used as a CLI tool to execute commands in a shell, but of course it is a full-blown programming language, and one that should be universally praised for its elegant design and high usability. We’re not going to talk about any Python language speciﬁcs here, but note that the full set of data types, variables, and built-in functionality and classes of the Python language is in fact an API for users to interact with. A lesser-known fact to beginners is that the Python API has many different implementations. When you say you’re programming Python, chances are that you’re referring to CPython, a ubiquitous, free, and open-source implementa- tion of the Python API based on the C programming language. However, there are alter- natives available, such as Jython (written in Java), PyPy (written in RPython), and IronPy- thon (written in C#). The take-away here is that while working with any given programming language it’s easy to forget that they’re all implementations of a design speciﬁcation, written in another programming language. Python is a very expressive language, but in the end it’s just an API for lower-level code. You can use this know- ledge to your advantage too, since working with CPython allows you to write extensions of Python written in C, which might just be faster than plain Python. Popular libraries such as numpy, a Python package for manipulating n-dimensional arrays efﬁciently, is by and large written as a C-extension to Python.

Programming libraries and frameworks

Stepping out another layer, the libraries and frameworks you’re using in practice—for instance libraries for data analyses or visualization—are themselves extensions of the programming languages they’re written in. We mentioned numpy in the last paragraph, which is an API for n-dimensional arrays expressed in Python. The key point is that you don’t have to think or reason about how numpy does what it does; you only need to understand the interface it is providing you with on a conceptual level. In other words, the interface is separated from the implementation details. This crucial principle underpinning all APIs is something we’ll investigate more closely in the next section. For now, let’s start with a simple example: Let’s say I want to create a 3-by-3 matrix in Python, add this matrix to itself, and then print the result. numpy allows you to do that as follows:

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

This returns the expected result

[[2. 2. 2.]

[2. 2. 2.]

[2. 2. 2.]]

API

The only thing you need to know is that the numpy API has a ones function call that creates matrices containing only the number 1 in all entries, of the speciﬁed shape (here 3 by 3), and that you can add matrices by using the “+” operator, which Python allows you to override. Crucially, you do not need to know how numpy stores matrices, or how it carries out matrix addition or any other operation. This decreases cognitive load and lets you focus on the use case you care about by only learning the API, not the implementation. Highly specialized libraries or frameworks can sometimes be cha- racterized as domain-speciﬁc languages (DSL) when they act as a de-facto standard and complete toolset for a concrete domain. Numpy is a DSL for n-dimensional arrays in Python, while pandas would be an example of a DSL for data frames in Python. Beco- ming an expert in any data-intensive science usually involves having the essential lib- raries for your domain under your belt.

Databases

Another important application domain that we already brieﬂy touched in the historical timeline of APIs is that of databases. Database management systems are complex soft- ware components that users must interact with, and each such system has languages, or APIs, that do precisely that. Modern databases created in the last one or two dec- ades deviate from the traditional way of doing things, but relational databases all share the common trade that they’re based on tables with precisely deﬁned column names and types. On top of that, relational databases all come equipped with a common query language called structured query language (SQL). There are variants and dialects of SQL, but fundamentally it is an API used to interact with relational databases: It is used to create, alter and delete tables, insert entries or modify them, retrieve and ﬁlter data, and so on. The internal structure of (relational) databases is fascinatingly com- plex and a lot of optimizations carefully hidden from the user go into modern database design. In fact, there are a lot of different ways to implement a relational database, inc- luding many different data structures and algorithms for data storage and retrieval to choose from, but to the user this is all a black box. All data access happens exclusively through the domain-speciﬁc language SQL.

Remote procedure calls

So far, we’ve only explicitly discussed APIs that relate to work on a single process on a single computer. However, it’s normal to have several processes running on a computer at practically all times. What’s more is that these processes have to communicate quite often: For instance, if you’re running a text processing tool to take notes and host a meeting with a conferencing software at the same time, your operating system is smart enough to copy a note from the former into the chat of the latter. That’s a form of inter- process communication (IPC). Going one step further, your colleague who sits at ano- ther computer entirely can read your note in his instance of the same conferencing tool running as a process on his computer. Letting computers “talk” to each other like this is a complicated topic that usually involves complicated communication protocols. An essential part of communication between several computers is networking, i.e., the way the computers are connected with each other, for instance over the internet. One spe- ciﬁc IPC paradigm between different machines, originating from the ﬁeld of distributed computing, is called remote procedure call (RPC), which is a request-response protocol. It means there are two parties involved: a client and a server, connected to each other via some network. The client sends a request to the server to let it execute a program

with parameters as speciﬁed. The server runs this program accordingly and returns the result to the caller, the client. The tricky part is that the program itself must be transfer- red over the network from client to server and the result must also be transferred from server to client. The complication stems from the fact that all in-memory objects of the client that are necessary for the function call have to be persisted ﬁrst, sent over the network, and then read into memory again by the server. Many RPC systems use an interface description language (IDC) to describe a contract, which can then be used to generate code for both client and server. A prominent and widely adopted RPC system is Google’s gRPC, which uses Google’s Protocol Buffer format as IDC and the web-wide standard hypertext transfer protocol (HTTP) for network transport. HTTP is one of the core protocols on which the World Wide Web heavily relies and powers most modern web APIs.

REST

The concept of representational state transfer (REST) was introduced in a PhD thesis in 2000 and speciﬁes a set of architectural constraints for web applications, like cli- ent-server architec- ture or statelessness of services, which helped web applica- tion developers to ﬁnd a common com- municational ground to build the massive, distributed network that is the internet.

Web interfaces

When you look at how rich and diverse modern web and mobile applications can be, it might strike you as surprising that almost all communication is backed by web proto- cols like HTTP. This protocol was ofﬁcially introduced in 1997. HTTP/2 has existed since 2015, and its successor HTTP/3 is already supported by some browsers (Berners-Lee, 1996). The modern web is based on a set of constraints or principles called representa- tional state transfer (REST), and applications that utilize this set of principles are often referred to as RESTful applications or services. APIs for RESTful applications are referred to as REST APIs, and often HTTP is used for communication in such interfaces. The evo- lution of the HTTP protocol as an API is interesting in its own right, but we’re going to focus solely on the semantics of HTTP and how you interact with it in practice. At its core, HTTP works with universal resource identiﬁers (URI), with which you can uniquely identify resources on the web, and operations or methods that tell HTTP what to do with the resource exactly. The basic process is that the client asks for an HTTP method to be executed, which the respective server corresponding to the URI does, after which the server then returns the result of the request to the client. The prototypical example is to type an address like google.com into your browser and hit enter. In this situation your browser is the client sending a GET request to Google’s servers, who respond with the HTML code for the landing page of Google, which in turn can be rendered by your browser. The full list of basic HTTP methods goes as follows:

* GET requests to get a speciﬁed resource and it is only used for data retrieval. A GET request has no other side effects, on the server or otherwise.
* HEAD is identical to GET, with the exception that you don’t receive any response body. For instance, when GET would give you the HTML code for a website, the corre- sponding HEAD request would only contain the meta information of the request (like information about the server), but no actual content. A minimal web applica- tion must specify at least HEAD and GET methods to qualify as such, all the other following methods are optional.
* POST differs from a GET request in that it comes with a request body that is posted to and accepted by the server and used to run the request before returning to the client. An example would be posting to a message forum, in which case the text of your post is contained in the request body.

API

* + PUT is somewhat related to POST in that it also sends content through the request body. However, it is semantically different because you require the server to store the body, i.e., put the sent resource there.
  + DELETE, unsurprisingly, deletes a previously stored resource, which you might have PUT there.
  + PATCH modiﬁes a previously stored resource, if only partially.
  + OPTIONS returns to the client the HTTP methods supported by the server.
  + TRACE is essentially there for debugging purposes and echoes incoming requests back to the client for it to check whether the original request has been modiﬁed or amended by the server.
  + CONNECT is method in which the client asks the server to be connected with ano- ther machine or through another protocol. In this situation the server acts as a proxy. This method is often used to set up a secure client-server connection through securely encrypted methods like HTTPS.

Graphical user interfaces

Lastly, graphical user interfaces (GUI) can also serve as a form of API. The operating sys- tem you’re using likely comes with a graphical interface and provides you with a gra- phical representation of the folder structure on your computer, a desktop, a mouse cur- sor, and other amenities. In the early days of computing all you had was a CLI, and in the very beginning not even that. Today GUIs are the default for most commercial applications running on desktop computers or smartphones. One particularly relevant application class of GUIs in the context of data-intensive sciences is that of integrated development environments (IDE), which give programmers a complete support system for their development efforts. Tools like PyCharm or Rodeo are popular choices for Python programmers and data scientists who want to be productive in their daily work. Such IDEs offer tooling for code completion, navigating complex projects, ﬁnding speci- ﬁc source code parts, debugging your code, or proﬁling the run-time performance of your programs. In a sense, IDEs provide you with an API for all things directly related to your programming tasks, and it is difﬁcult to imagine such powerful tools without the graphical interface.

### API Design Principles

You’ve now seen the many forms in which APIs can come and what they look like. It’s time to investigate how to build APIs in practice and get a feeling for good and bad designs by understanding some core principles of API design. Simply put, when writing anything, and especially code, you want to be understood. To start out, let us have a look at two concrete examples of machine learning software workﬂows. The ﬁrst one is an example using Ray, an up-and-coming distributed computing framework for Python that supports a variety of algorithms for a machine learning paradigm called reinforce- ment learning. You install this Python library with pip install ‘ray[rllib]’ and to check if the installation worked you could run

import ray ray.init()

HTTPS

An extension of HTTP, HTTPS uses a cryptographic proto- col-like transport layer security (TLS) or its predecessor secure socket layer (SSL) to secure the connection between clients and servers and to prevent frau- dulent behavior on the internet.

in a Python session. That should result in no errors, so we should be able to run a ﬁrst example by typing the following line in your shell:

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

When you do so, you will notice that this program does not run correctly; it gets inter- rupted by an error message embedded in an output that’s simply too long to put here in its entirety. After sifting through about 80 lines of code you eventually ﬁnd a clue:

ImportError: Could not import tensorflow

The reason is that this Ray example relies on Google’s TensorFlow library, which you’d have to install ﬁrst. The problem with this example is not that Ray didn’t install Tensor- Flow, which in fact it shouldn’t. It’s that we veriﬁed our installation, but Ray didn’t com- plain; it happily accepted our setup and started running the example. What’s more, the error message is too difﬁcult to read and novices have a hard time extracting informa- tion from such long messages. The last reason why this example is problematic is that the error message occurred too late in the program ﬂow. The error occurs after Ray has set up its distributed computing framework, which is why you ﬁnd the relevant error message about TensorFlow only after about 50 lines of program output. Now, the rea- son we’re discussing this example at great length is that it’s an example of bad API design. Imagine you run a complicated machine learning experiment with hundreds of lines of code. You start the experiment in the evening to see the results in the morning, only to notice that you forgot to install TensorFlow and the program crashed right away. That can mean a huge dip in productivity and it all comes down to a questionable design choice in error handling. In contrast, let’s see how a different machine learning framework handles the exact same situation. This time we’re looking at Keras, a popu- lar library for deep neural networks. As before with Ray, it’s not necessary for you to know anything about machine learning, we’re just concerned with API design in this section. After installing Keras with pip install keras, you can validate the installa- tion by typing

import keras

into an interactive Python session. What you’ll see right away is the following error message:

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

Note how Keras not only tells you immediately that something went wrong, it also tells you what exactly it requires and suggests a concrete action to solve the problem in human-readable form without searching through a massive wall of text output. This type of error handling demonstrates excellent API design and goes to show that well- written APIs can save you a lot of problems down the road. A good API improves devel- oper experience and enhances productivity. Keras has been hailed for its API design since its inception in 2015 and in recent years understands itself less as a concrete library and more as an API speciﬁcation that you can implement and follow (Sadrach,

API

2020). Since the release of TensorFlow 2.0, previously a standalone library, Keras is now an ofﬁcially supported API of TensorFlow (Chollet, 2017). We’ll get back to Keras in this section to illustrate a few more points about good design.

###### The Zen of Python

One of the reasons the Python programming language is recognized as elegant and easy to start with is that it has been built with a design philosophy in mind that has been eternalized in the so called “Zen of Python.” The Zen is a list of ﬂexible principles, not to be confused with a rule set, which you can read using any Python version by typ- ing import this into a Python session. We’ll go through these principles in detail, as they make some good points about API design and its difﬁculties. On a side note, this philosophy has been lovingly mocked by Daniel Greenﬁeld with an Anti-Zen library called that, which shows you anti-patterns to avoid. An aggregated view on the Zen of Python reads as follows:

* + Beautiful is better than ugly. Python wants you to prefer beautiful solutions over ugly ones, which can be highly subjective. Working with intuitive concepts that click has merits for your users.
  + Explicit is better than implicit. Hiding information from your users might require them to know about your implicit rules, so be as explicit as possible.
  + Simple is better than complex, complex is better than complicated. The Python equivalent of Occam’s razor tells you to shed complexity whenever possible. If you have to incur complexity, at least make sure not to complicate things.
  + Flat is better than nested, sparse is better than dense. Prefer data structures and concepts that aren’t too entangled and nested. For instance, a ﬂat Python dictionary with properties is usually preferred to a nested one. Also, don’t cram too many things into one line of code in order to avoid too much density.
  + Readability counts. Needless to say, code that’s easier to follow is more fun to read and easier to maintain long-term.
  + Special cases aren't special enough to break the rules, although practicality beats purity. This is a very interesting part of the Zen, since it shows you on a meta level that you have to take these principles with a grain of salt. While you should not usu- ally let special cases ruin your overall design, that is exactly what might be neces- sary sometimes (representing an exception to the rule).
  + Errors should never pass silently, unless explicitly silenced. When errors occur fast and quickly, the user never has to guess what the reason for a program failure might be. Doing so might make your program ﬂow more complicated but it is a good design choice. In rare cases, when it’s safe to do so, error messages can be silenced.
  + In the face of ambiguity, refuse the temptation to guess. In other words, try to avoid ambiguity by being as explicit and clear as possible.
  + There should be one—and preferably only one—obvious way to do it, although that way may not be obvious at ﬁrst unless you’re Dutch: This reference on the Dutch creator of Python Guido Van Rossum is probably the most controversial of all the principles on this list. Python programmers refer to this way of doing things as “pythonic.” Often there’s no doubt about a pythonic solution and the community

largely agrees on certain patterns, but sometimes it’s just not clear if one approach is actually better than another. Still, the point is to strive for code that’s so obvious to use that you can’t go wrong.

* + - Now is better than never, although never is often better than right now. When designing things, don’t delay your process and ﬁnd a solution now, even if it is sub- optimal. You can iterate and improve on it later. However, if hard-pressed, don’t go down a road that makes things too difﬁcult to ﬁx later on.
    - If the implementation is hard to explain, it’s a bad idea; if the implementation is easy to explain, it may be a good idea. This slightly pessimistic view on programming yet again alludes to beauty and understandability. Not everything that’s easy to explain is good, but if you can’t explain it, it’s certainly bad.
    - Namespaces are one honking great idea—let’s do more of those! Python makes ample use of namespaces to modularize and structure its code. Let’s say you imple- ment a zoo with plenty of animals in Python. Using proper name-spacing, your Ele- phant class might be imported as from zoo.animals.mammals import Elephant, instead of from zoo import Elephant, as the latter might clutter your namespace with too many concepts.

###### Separating Interfaces and Implementations

Separation of con-

cerns In software engi- neering, a common principle is to sepa- rate concerns, which means that you build your software in such a way that each part or module is solely responsible for certain aspects and concerns should never be divided by two or more such

modules.

A common principle in interface design is to separate interfaces and their implementa- tions. Strictly speaking, by the deﬁnition we’ve given and the properties discussed, an API is simply an interface, and the whole point is that its users don’t have to care about implementation details. In practice, however, it’s good to be reminded of this principle sometimes. Interface separation can be broadly understood within the principle of sep- aration of concerns. Strongly typed languages like Java or Scala usually have a dedica- ted class for interfaces, which enforce this practice. In Java, an “interface” can’t have a concrete implementation, as it is prohibited by design. If you want to work with func- tionality as laid out in the interface, you have to provide an implementation of the interface by writing a subclass for it. On a more fundamental level, the two program- ming languages Java and Scala, which are syntactically quite different, compile to com- patible byte code on the Java virtual machine (JVM). In particular, you can import Java classes into your Scala code and build software on top of it. In Python, there is no such concept as an interface in the basic language, and a strict separation of interfaces and concrete implementations are considered an anti-pattern (or un-pythonic). However, that does not completely invalidate the principle just presented. In Python, you still want to carefully think about the objects your users will interact with, the public API of your code, and the rest of the code, the implementation. If you view the public-facing API as a “surface area” which most of your users will interact with, for larger projects this is really just the tip of the iceberg, which makes good interface design even more important.

API

###### Abstractions

When building APIs, an important aspect is to ﬁnd the right abstractions, i.e., the right concepts that go well with what you intend to build. Abstractions should feel natural and make it easy for your users to interact with them. As famous computer scientist Edsger Dijkstra puts it, “[b]eing abstract is something profoundly different from being vague... The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise” (Misa & Phillip, 2010, p. 1). To give you one example, the Keras library gets away with very few high-level concepts. To illustrate the elegance of Keras’ approach you need to know that deep neural networks, which Keras is built for, are machine learning models built from layers of relatively simple compute instructions. The simplest models are sequential in nature, which means that the layers involved follow one after another. The simplest layer in a neural network is probably a densely connected layer, meaning that all neurons (a term borrowed from neuroscience used to refer to how the brain works) in one layer are connected to that of the next. Even if you’ve never heard about deep neural networks before, now that you’re equip- ped with this knowledge, you can read the following Keras model speciﬁcation:

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))

In other words, to construct a sequential network with Keras, you sequentially add as many layers as you would like to. The speciﬁcs of this model are irrelevant here, just note that it almost reads like plain English and works with precisely the same high- level abstraction we just explained: models and layers. You might say that this is an obvious thing to do, but far from it: Before Keras, none of the deep learning frameworks adopted a similarly simple and intuitive style of programming, which means the discip- line of deep learning was much less accessible to practitioners. In 2020, the landscape has changed for the better, but the impact of well-written APIs such as Keras on the hype of deep learning and machine learning in general is not to be underestimated. It is famously said that all abstractions are leaky (Spolsky, 2002), meaning that no matter how well you think through an abstraction in software engineering, there will be cases at which you will have to look “under the covers” or break the rules for special cases. This reﬂects the fact that the world can be messy, and abstractions are just an attempt at grouping things into categories, a process which can fail for several reasons. While that statement is true, it is also often irrelevant in practice to the extent that good abstractions go a long way, while poor choices might make you fail badly. Another way to reason about abstractions, especially when it comes to leakiness, is that you might require different levels of abstraction. Sometimes you’re doing just ﬁne with the hig- hest level of abstractions, but sometimes you have to go a few levels deeper to adjust code to your use case.

###### Error Handling

We’ve already seen good and bad error handling in an earlier comparison of the Ray and Keras Python libraries, but it’s worth generalizing and summarizing a few points here that are important to keep in mind:

* + - When throwing an error, fail as early and quickly as possible. You do this for maxi- mum transparency and to prevent more serious errors down the line, which might be more difﬁcult to understand and ﬁx.
    - Your error messages should be human-readable and clear. They should not only give the user a local error message that is hard to interpret, but they should also give the context in which the error occurred and what the user can do to ﬁx it.
    - Provide users with messages at different levels, for instance, provide useful warn- ings about a deprecated function that might be removed from the next release.

###### Convention and Conﬁguration

According to computer scientist Phil Karlton, “There are only two hard things in Compu- ter Science: cache invalidation and naming things” (Fowler, 2009, p. 1). While this is a bit tongue-in-cheek, there’s a lot of truth to that statement. When it comes to API design, there’s mostly just one part of the quote that is relevant: naming things. Finding good naming conventions is surprisingly hard and some languages, like Java, have a cultural tendency to have very long class names. Take AbstractTransactionalDataSource- SpringContextTests from the Spring framework, for example. That is a great example in the sense that it should be avoided, and likely points to either a conceptual lack of clarity in your abstraction or the fact that your class simply does too much (and hence violates the separation of concerns principle). Giving clear and precise names to your classes and functions should be something you always keep in mind when designing software. Apart from naming things, other types of conventions are important, too. For instance, in Python you can have default arguments in your function signatures. You should always make good use of them, whenever you have a sensible default to put there. Most of your users won’t be power users and don’t care for every single parame- ter that you can tweak. Having said that, it’s better to have a lot of parameters in your function parameters that your users can conﬁgure, than it is to let them modify them through code. To give a concrete Python example, let’s say we have a simple class encapsulating a bucket, which has a size and can potentially be ﬁlled with something. Here’s a Python class for that:

class Bucket:

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

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

API

The default bucket size is 10 (the meaning or unit of this number is unimportant here), so users can create a bucket using this convention with

bucket = Bucket()

They can also conﬁgure the bucket size upon initialization using

bucket = Bucket(size=20)

And ﬁnally, users could also ﬁrst create a bucket and then modify its size through code (a questionable operation altogether):

bucket = Bucket() bucket.set\_size(20)

A good rule of thumb to follow is “convention over conﬁguration over code.” Provide as many good default values as possible and make them easily conﬁgurable.

###### User Experience

A lot of the principles discussed in this section come down to applying common sense reasoning to the domain of software engineering. Ultimately that’s because building good APIs, or designing things in general, is all about empathizing with your users and respecting the cognitive factors that come into play when building things for humans. In other words, user experience, also called developer experience when applied speciﬁ- cally to the relationship of engineers to your code, is important. The cognitive dimen- sions framework (Clarke, 2004) lists a total of 12 points in relation to how software engi- neers work with APIs and what they expect from them.

* + Abstraction level: What’s the maximum and the minimum number of abstractions found in your API, and what comes in-between? What level should your users work with primarily?
  + Learning style: What are the requirements for learning the API? Does it have a steep learning curve? What’s the target group of developers, and what’s their background?
  + Working framework: What’s the cognitive load or conceptual chunk needed to understand this API and work with it effectively?
  + Work-step unit: How much work is required by the user in each step? Is the API par- ticularly dense in that regard?
  + Progressive evaluation: Can you build up code parts progressively and get feedback on your progress? Or do you need, in extreme cases, to bring all pieces together to see if an idea works?
  + Premature commitment: If I go down one path, how many other non-trivial follow- up decisions do I have to keep in mind as a consequence of this decision?
  + Penetrability: How well can you explore the API on your own? Is it well documented and can you understand and compose its components intuitively?
  + API elaboration: How tailored to your needs is the API? Are there possibilities to ﬂex- ibly conﬁgure it, or do you have to modify it heavily to suit your needs?
    - API viscosity: Is it easy to modify or extend the API for a different use case? Does it speak the language of the developer community it targets?
    - Consistency: If you’ve seen one part of the API, can you reason about the workings of a different yet related part of it? In other words, is the API consistent in its choice of building blocks?
    - Role expressiveness: Do the abstractions picked by the API match your cognitive model of the components as they relate to the program ﬂow overall?
    - Domain correspondence: All APIs are written within a context, for a speciﬁc domain. How well does the API at hand express the models present in that domain?

###### Dokumentation

Lastly, good documentation is crucial for any successful API with a large user base. Make it a habit to

* + - document your code.
    - document your interfaces, input parameters, and return values.
    - document the intended use of your API.

### Building a Python Library

Having looked at various API paradigms and design principles, it’s now time to build a concrete Python library to apply what you’ve learned. The focus of this library is not so much about the implementation details of the library as it is about its API. In fact, the code we’re writing behind the API will be relatively complicated in order to show you how good API design can hide it. To offer a concrete example, let’s say you run a com- pany specializing in analyzing large corpuses of text documents, and you sit on a pro- prietary algorithm that can do large-scale analyses faster than everyone else. Of course, you want to bring that algorithm in the form of a service to your users, to pro- vide value and make money. The ﬁrst step for you is to write a Python library that prop- erly encapsulates your algorithm and makes it accessible for users, without sharing your secret algorithm. To restrict things even further, we only do one thing: count all occurrences of words across several documents.

###### An Algorithm to Summarize Documents

Let’s start with the implementation of the core algorithm ﬁrst. This is a toy version of a programming paradigm called MapReduce, which is a milestone in distributed compu- ting and one of the many impressive contributions to large-scale computing by Google (Dean & Ghemawat, 2004). Many commercially successful big data technologies like Hadoop are based on this programming model. It is roughly based on three steps:

API

1. Take a set of documents and transform or “map” the elements you’re interested in (for instance, the words of the document) according to a function you provide. This step returns key-value pairs, in which the keys are the document elements you care about and the value is an entity you want to compute. In our use case we’re interes- ted in counting words, so each word x in a document would result in the key-value pair (x, 1), where the “1” signiﬁes that we counted the word once.
2. Collect and group all the output pairs from the last step. Let’s say the word x appea- red four times throughout the documents. The grouping step would then result in the value x: [1, 1, 1, 1].
3. The last step is to aggregate or “reduce” the elements from the last stage. In our case, we just want to sum all values for a total count. In the example of the word x, we would therefore get the result x: 4.

As you might guess, the paradigm MapReduce gets its name from stages 1 and 3, but the second one is important as well. While the three steps may look simple, their power lies in the fact that they can be massively parallelized across hundreds of machines. Let’s provide a naïve implementation of this algorithm for our use case, word counting, in Python. We start with implementations of the three phases. First, we deﬁne a map function that returns (word, 1) as key-value pair for each word in the text:

def map\_function(text):

for word in text.lower().split():

yield word, 1

Then, we apply the map function to all elements in our data set data (a list of text data):

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

To group the key-value pairs produced by the map step by their key, we do the follo- wing:

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

Finally, we deﬁne a reduce function that sums up the counts of words:

def reduce\_function(key, values): total = 0

for count in values: total += count

return key, total

As with the map function, we also need to apply the reduce function to all key-value pairs from the group phase:

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

To see how this all works, let’s create a test data set to run this program:

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]

When we now apply all three MapReduce stages sequentially, we get the following out- put:

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 ...}

Now that we’ve checked that our code works as intended, let’s take a step back. We’ve implemented a toy version of a powerful, commercially successful algorithm and applied it to a concrete use case. However, we do not have an API for this algorithm yet. The last thing we want to do is share our intellectual property, so we need to carefully hide all implementation details and provide users with a simple interface to analyze their documents. How can we do that? First, we have to clarify the speciﬁcations of the API by understanding what our users might want from us. Let’s say we create a mini- mum viable product (MVP) by constructing the following Python functionality that we can share with users:

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

Of course, to protect our IP, users will only have access to count\_words\_naive, not its implementation. After an initial round of interviews with potential customers we get the following feedback which we can use to reﬁne our approach:

* + Many users didn’t understand how to use this function. The input argument corpus

is not documented and it’s not clear how to use it.

* + Some users ﬁgured out that our API takes a list of Python strings, representing the text documents to analyze, but had the problem that the function sometimes throws obscure errors, and they didn’t know why.
  + Users who got the result sometimes wanted something slightly different: They were interested in words with a minimum number of occurrences in the corpus (other- wise the output is too large).
  + Some users were not proﬁcient enough in Python and just wanted to provide a list of ﬁles, instead of loading documents into Python themselves ﬁrst.

To incorporate this feedback, we ﬁrst design a new backend function to ﬁlter words whose count is too low, which looks as follows:

def filter\_function(results, min\_occurrences=4): return {k: v for k, v in results.items() \

if v >= min\_occurrences}

With this ﬁlter function, we can now create a much-improved interface by taking several design principles from earlier into account, namely, proper and intuitive error handling, good documentation, using good conventions and conﬁgurations, and even making use of Python’s type checking module typing for input parameters and return types:

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)

Note that, while this might look like much more than we had in our naïve deﬁnition, most of the additional lines of code go into type checking, documentation, and cat- ching errors. That is a good sign and mature interfaces tend to look this way. To address the need for an API call that works with ﬁles instead of in-memory strings, we can deﬁne a new interface that calls the above count\_words function internally. Reusing function calls instead of creating separate code bases is another good design choice.

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

There is a lot to be unpacked in this piece of code. First, the arguments of count\_words\_from\_files are different from count\_words in many ways. The “ﬁles” parameter still takes a list of Python strings, but the meaning is different now. As the documentation explains, “ﬁles” is now a list of paths on your local system to ﬁles that should be analyzed. The “ﬁlter” parameter is the same as before, but we have a new parameter: skip\_corrupted\_files. This parameter accounts for the fact that not every ﬁle on your system can be opened and contains text data. skip\_corrupted\_files is false by default, which means that the program will stop immediately when it ﬁnds a

API

ﬁle that can’t be read. This is a smart default, as it’s more secure than just skipping such an event, but users might want to turn this feature off at their own risk. Hence, we expose this parameter to allow users to conﬁgure it as they need. As for the code itself, the ﬂow is straightforward. We iterate over all ﬁles and try to read them, and if we suc- ceed, we pass the result into the count\_words function deﬁned before.

Zusammenfassung

In this unit you’ve learned the fundamentals of application programming interfaces (APIs), their design, and their implementation to get you started with data-intensive science projects on that subject. All programming examples in this unit were given in Python, as it is easy to prototype with and currently the most relevant language when it comes to data science and related ﬁelds. The ﬁrst section exposed you to the basic terminology of APIs and how various levels of abstraction, from operating systems and hardware access to modern web interfaces, come into play when work- ing productively with a computer. These layers of APIs span a wide class of impor- tant entry points for application development and include database access, pro- gramming languages, and the use of your preferred toolchain in data-intensive ﬁelds. You now understand APIs as speciﬁcations between the API designer and user, a binding contract that parties must adhere to. The focus of this section was on the usage of libraries and frameworks, as well as the most important building blocks of RESTful web applications. Next, we had a closer look at what it means to actively build APIs, by introducing key design principles, dos and don’ts. You’ve learned about the importance of separation of concerns, good error handling, sufﬁ- cient documentation, ﬁnding good abstractions that aren’t too leaky, and using conventions and conﬁgurations to help users navigate your APIs in practice. The following section gave you a concrete use case of a small Python library built to share with your users. We built out a relatively complex underlying algorithm to count words in documents and focused on building a minimal user-facing API to expose it. You’ve seen how to effectively hide functionality that’s not necessary for your users to know, document your code unambiguously, use type hints to improve validation of input and output arguments, and handle errors properly in order to catch unintended behavior early on. To conclude, you can now identify strengths and weaknesses of APIs when you’re working with them, build them effectively, and set up your own libraries and REST interfaces in Python.



# Lerneinheit 5

## From Model to Production

#### LERNZIELE

Nach Abschluss dieser Lerneinheit sind Sie in der Lage...

… 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. From Model to Production

### 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,

From Model to Production

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.

From Model to Production

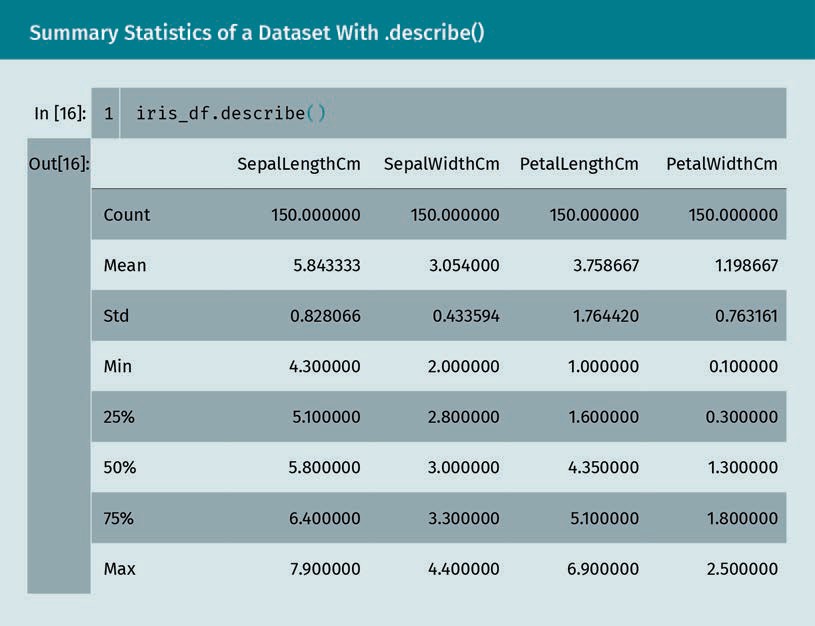
It is an intuitive process that requires experience, so we can only give a simple example of it. We will perform a bit of exploration of the iris dataset, a small dataset containing three classes of 50 instances each, where each class refers to a type of iris plant and is often used for demonstration 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.

From Model to Production

* + 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

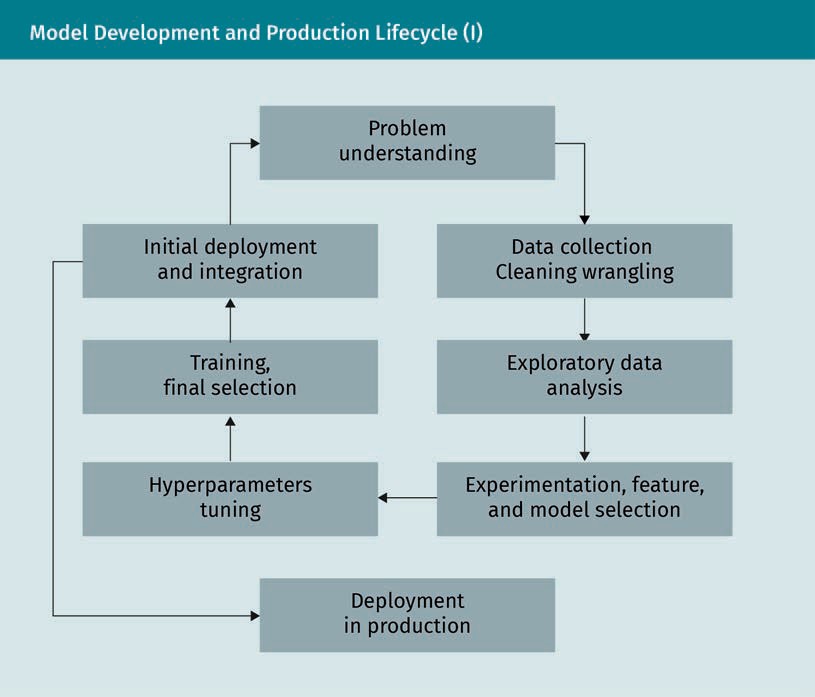
dataset is used for a ﬁnal unbiased evalu- ation of the per- formance.

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

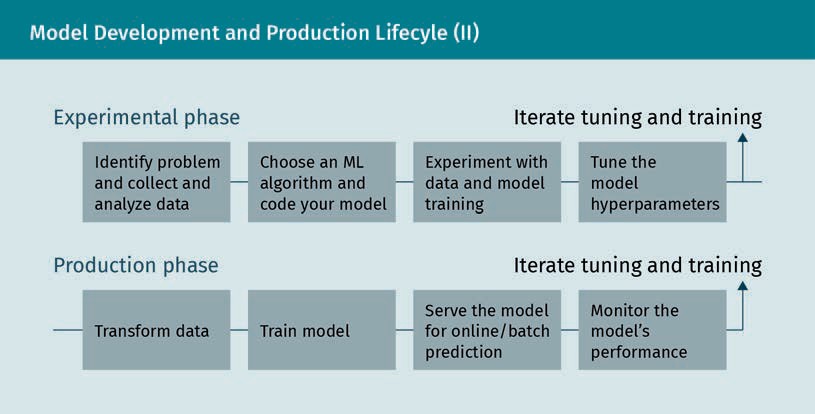
Let’s compare the traditional software engineering lifecycle to the machine learning lifecycle. In both cases it is necessary to understand the problem to be able to solve it correctly. However, in the traditional software world, the problem is often better deﬁned, at least at the level of pure implementation.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

From Model to Production

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.

###### Version Control

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.

From Model to Production

###### 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.

From Model to Production

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.

From Model to Production

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.

From Model to Production

###### 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

From Model to Production

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)

From Model to Production

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

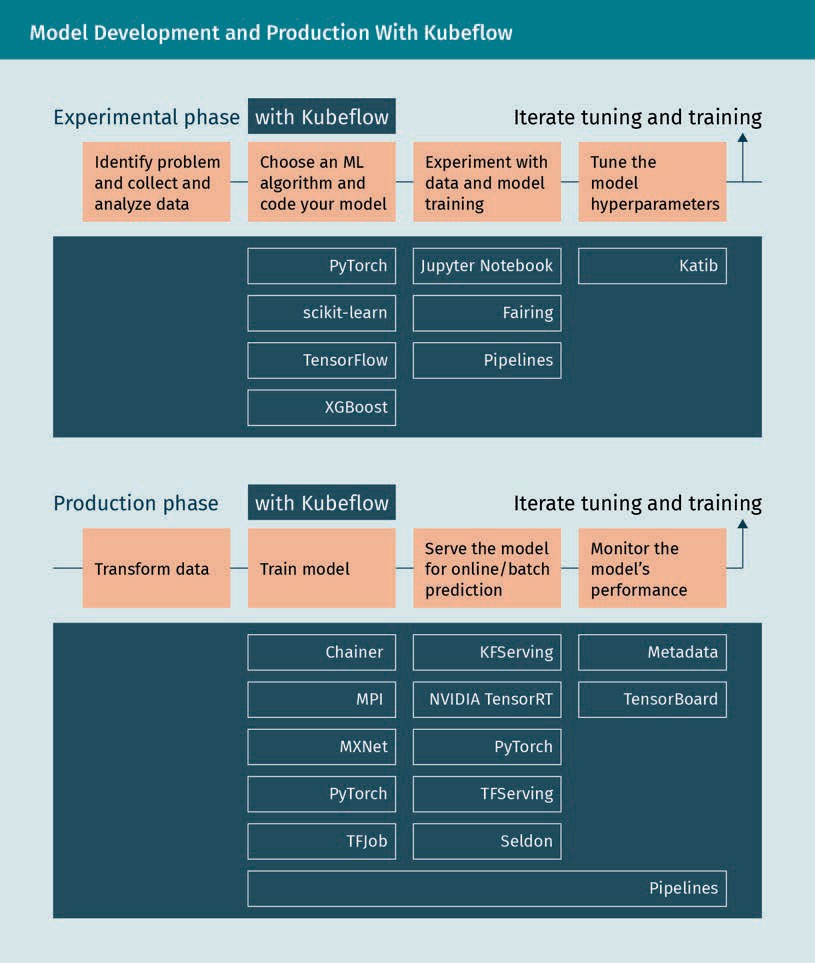
Kubeﬂow was started by Google as an open-source platform for running TensorFlow. It is a machine learning platform that manages deployments of ML workﬂows on Kuber- netes and is a scalable and portable solution (Kubeﬂow. n.d.-a). It is aimed at data scientists building and experimenting with data pipelines, 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

From Model to Production

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

From Model to Production

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.

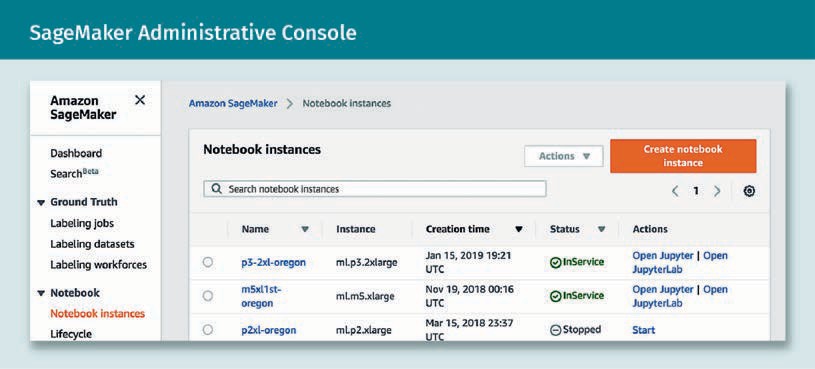
Data lake A data lake is a sys- tem or repository of data stored in its natural (raw) format, usually object blobs

or ﬁles.

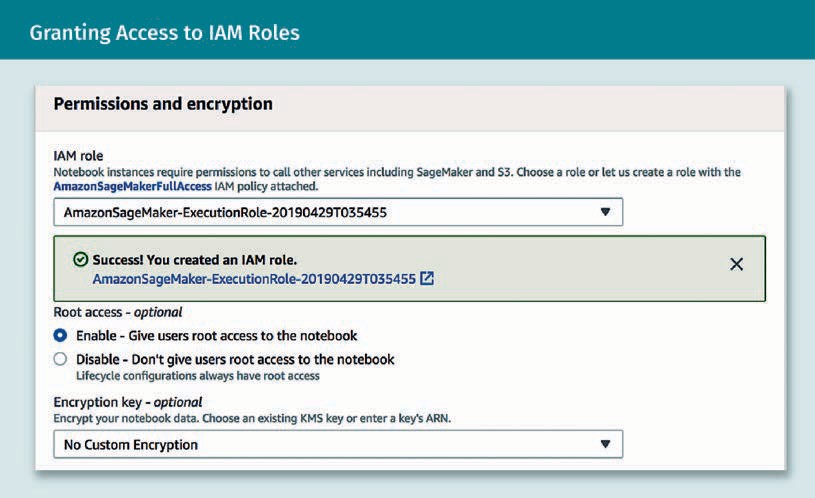
Data collection, cleaning and wrangling

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.

From Model to Production



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-

From Model to Production

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

From Model to Production

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-> 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.micro- soft.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